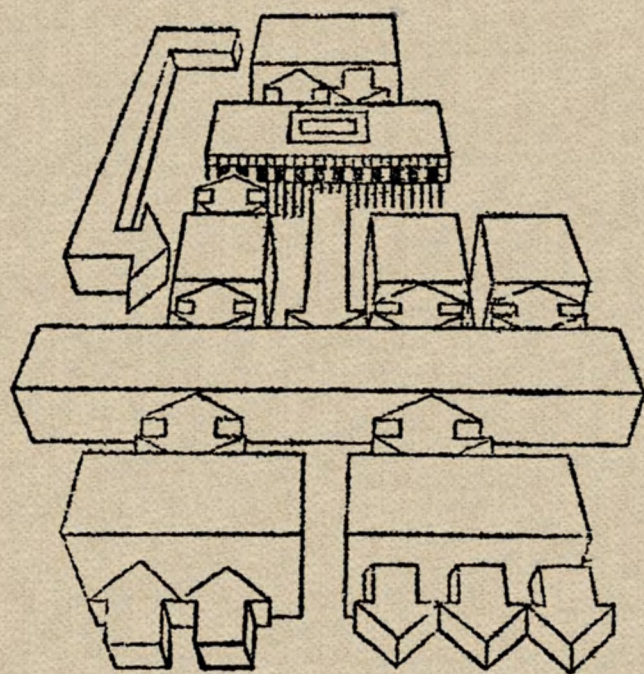


Микро-ЭВМ

Микро-ЭВМ



Микро-ЭВМ

Под редакцией А. Дирксена

Перевод с английского
под редакцией
В. В. Сташина



МОСКВА
ЭНЕРГОИЗДАТ
1982



Scan AAW

ББК 32.97
М 59
УДК 681.32—181.48

Рецензент И. В. Прангишвили

MICROCOMPUTERS

EDITED BY A. J. DIRKSEN

Kluwer technische boeken by Deventer — Antwerpen, 1979

Микро-ЭВМ /Пер. с англ. под ред. А. Дирксе-
М 59на. — М.: Энергоиздат, 1982. 328 с., ил.

В пер.: 1 р. 50 к.

В книге голландского специалиста излагаются общие принципы организации микропроцессоров и микро-ЭВМ. Описываются специфика работы и особенности программного обеспечения микро-ЭВМ на базе микропроцессора Intel 8080.

Книгу отличает продуманная методика и простота изложения сложных вопросов теории цифровых устройств и методов цифровой обработки данных.

Для специалистов, работающих в области вычислительной и измерительной техники, автоматики, телемеханики. Может служить учебным пособием студентам соответствующих специальностей.

М 2405000000-150
051(01)-82 230-82.

ББК 32.97
6Ф7

© 1979 Kluwer Technische Boeken B. V. — Deventer

© Перевод на русский язык, Энергоиздат, 1982

ПРЕДИСЛОВИЕ РЕДАКТОРА ПЕРЕВОДА

Микропроцессоры (МП) и микро-ЭВМ благодаря реализованной в них возможности программного управления обладают свойствами универсальных устройств и позволяют применять средства и методы цифровой обработки данных и цифрового управления в таких областях техники и народного хозяйства, в которых ранее их использование было экономически неоправданным.

Постоянно возрастающий объем производства МП и микро-ЭВМ, улучшение их технических характеристик и снижение стоимости дают возможность утверждать, что в ближайшие годы МП и микро-ЭВМ будут очень широко использоваться в устройствах и системах автоматики, телемеханики и связи. Опыт применения МП и микро-ЭВМ в локальных устройствах автоматики и системах управления объектами и технологическими процессами свидетельствует о том, что использование МП обеспечивает достижение исключительно высоких технико-экономических показателей и расширение функциональных возможностей этих устройств и систем.

Построение устройств и систем автоматики на основе программно-управляемых компонентов (МП и микро-ЭВМ) предопределяет не только применение принципиально новых структурных решений и новой организации процессов управления объектами, при которых в полной мере используются функционально-логические возможности МП, но и в значительной степени изменяет характер труда разработчика. Средства и методы проектирования устройств и систем автоматики на основе МП и микро-ЭВМ перемещают основной объем и содержание проектных работ из области схемотехнического проектирования в область процедурного программирования. Другими словами, с использованием МП и микро-ЭВМ традиционная задача управления объектами и процессами решается

не на основе схемной реализации управляющего автомата, а путем программирования процедур сбора и обработки данных, программирования процедур формирования и выдачи управляющих воздействий, программирования процедур контроля и диагностики и т. д. Эта принципиально иная методология проектирования устройств и систем автоматики на основе МП и микро-ЭВМ потребовала от разработчиков аппаратуры овладения средствами и методами вычислительной техники.

Данная книга, которая представляет собой учебник, ориентированный на инженеров по автоматике, поможет читателю овладеть основами цифровой техники, арифметическими и логическими принципами организации МП и микро-ЭВМ, а также методикой разработки программ для решения прикладных задач пользователя. Такие главы книги, как «Система адресации», «Синтаксис и подпрограммы», «Блок-схемы алгоритмов», «От постановки задачи к решению» и «Примеры простых программ» предназначены для читателей, которые не имеют практического опыта программирования для микро-ЭВМ.

Книгу отличает простота изложения сложных вопросов теории цифровых устройств, удачная последовательность изложения материала, значительное число примеров программ, хорошо иллюстрированное описание средств и методов проектирования цифровых устройств автоматики на основе МП и микро-ЭВМ. Продуманная методика описания процесса разработки устройств с программным управлением поможет читателю не только осознать те коренные изменения, которые произошли в области проектирования устройств автоматики в связи с появлением МП и микро-ЭВМ, но и позволит инженеру, не специализировавшемуся ранее в области вычислительной техники, приобрести первоначальные знания и навыки, необходимые для применения МП и микро-ЭВМ в устройствах и системах управления объектами и процессами. Значительный интерес для советского читателя представляет и то обстоятельство, что авторы книги в качестве базового выбрали широко распространенный микропроцессор Intel 8080. Описание методов и приемов конструирования программ, реализующих типовые процедуры и функции цифровых устройств, авторы приво-

дят с использованием языковых выразительных средств этого типа МП.

Перечисленные достоинства книги позволяют рекомендовать ее широкому кругу специалистов для ознакомления с техникой МП и вспомогательными средствами проектирования систем на их основе. Книга может быть с успехом использована студентами, специализирующимися в области проектирования, применения и эксплуатации средств вычислительной техники в системах сбора и обработки данных, а также в устройствах автоматики, телемеханики и связи.

Главы 1, 7, 14, 18, 19 и 20 переведены на русский язык канд. техн. наук, доц. В. Г. Черновым, гл. 2, 3, 6, 12 и 13 — ст. науч. сотр. К. В. Мышаком, гл. 4, 5, 8, 9, 10, 11, 15, 16, 21 и приложение — канд. техн. наук, доц. В. Ф. Кочневом, гл. 17 — редактором, канд. техн. наук В. В. Сташиным.

В. В. Сташин

Глава первая ЧТО ТАКОЕ ЭВМ

1.1. Введение

Электронно-вычислительная машина (ЭВМ) представляет собой устройство, предназначенное для выполнения вычислительных операций.

Однако помимо вычислений ЭВМ может выполнять операции, которые на первый взгляд не имеют ничего общего с вычислениями. Примерами таких операций могут служить перевод и редактирование текстов, передача данных, бухгалтерский учет, управление производственными процессами и т. п.

В рамках данной книги будем понимать под ЭВМ устройство, в котором обработка данных осуществляется под управлением программы. Обработываемые в ЭВМ данные представляют собой двоичные коды. Двоичные коды (совокупности нулей и единиц) преобразуются в электрические сигналы. Таким образом, ЭВМ представляет собой электронную цифровую вычислительную машину.

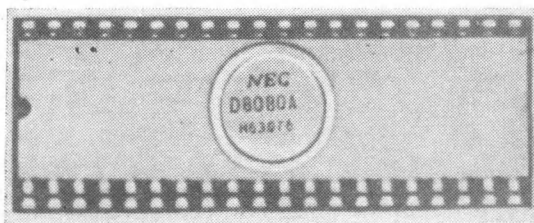


Рис. 1.1.

Термином *микро-ЭВМ* принято обозначать такие ЭВМ, в которых большая часть электронных схем размещена в одной интегральной микросхеме (ИС). Они отличаются от обычных ЭВМ только размерами и стоимостью. Типичная интегральная микросхема (рис. 1.1) имеет геометрические размеры $4,5 \times 1,5$ см². Сам кристалл, в котором расположены электронные схемы, имеет площадь 1 см².

Совокупность электронных схем выполняет над поступающими в них данными ряд операций обработки. Поскольку все эти обрабатываемые схемы размещены на поверхности очень небольшой ИС, то она называется *микропроцессором*. Стоимость микропроцессоров настолько мала, что, дополнив микропроцессор запоминающим устройством и устройствами ввода-вывода, можно получить недорогую микро-ЭВМ. Целью данной книги является изложение основных принципов использования микро-ЭВМ. Эти принципы касаются как вопро-

сов организации *аппаратных средств*, так и вопросов организации *программного обеспечения*. В данной главе рассмотрим структуру ЭВМ и принципы программирования ЭВМ.

1.2. ЭВМ как электронное устройство

Структурная схема, приведенная на рис. 1.2, а, пригодна для описания принципов работы любого электронного устройства обра-

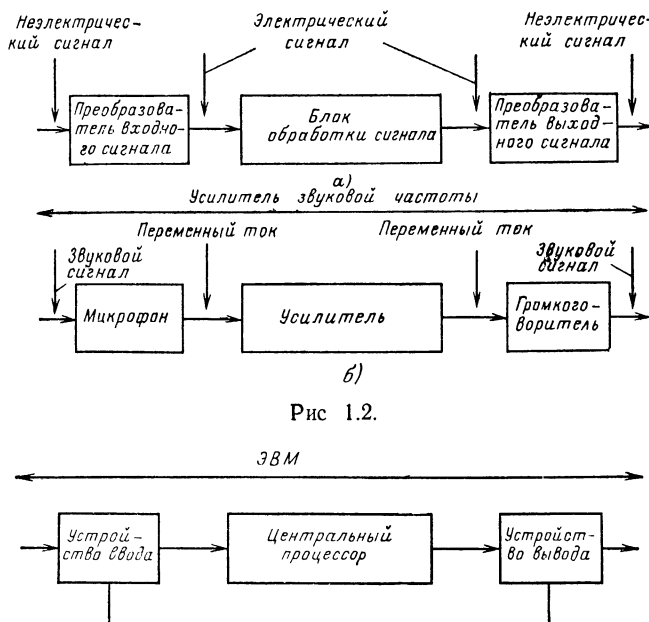


Рис 1.2.

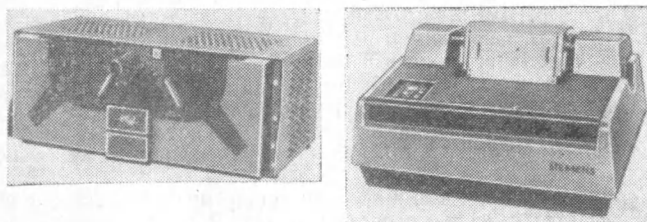


Рис. 1.3.

ботки данных. Преобразователь входного сигнала трансформирует поступающую информацию в электрический сигнал, который затем

обрабатывается в блоке обработки, построенном на электронных схемах. Затем обработанный сигнал преобразуется в незлектрический сигнал с помощью преобразователя выходного сигнала. На рис. 1.2, б показана принципиальная схема работы усилителя звуковой частоты, который использует те же принципы, которым отвечает система, показанная на рис. 1.2, а. Блок обработки сигнала здесь называется усилителем. Микрофон выполняет роль преобразователя входного сигнала, который преобразует поступающую информацию (звуковой сигнал) в электрический сигнал (переменный ток). Громкоговоритель выполняет обратное преобразование.

Соответствующие блоки ЭВМ показаны на рис. 1.3. *Устройство ввода* преобразует входные данные; например, *устройство ввода с перфокарт* преобразует коды, образованные пробивками на перфокарте, в совокупность электрических сигналов. Эти данные обрабатываются в *центральной процессоре* (ЦП).

Устройство вывода (например, построчно печатающее устройство) осуществляет обратное преобразование электрических сигналов в текст. Совокупность устройств ввода и вывода данных принято называть устройствами ввода-вывода.

Выводы

1. Электронная вычислительная машина — это устройство, которое обрабатывает данные под управлением программы.

2. Работа ЭВМ подобна работе других электронных систем. Как следует из принципиальной схемы, в ЭВМ имеются преобразователь входного сигнала, блок обработки сигнала, центральный процессор и преобразователь выходного сигнала.

3. Микро-ЭВМ — это ЭВМ, которая реализована на элементах, расположенных в одной интегральной микросхеме. Подобная ИС называется микропроцессором.

1.3. Как осуществляется обработка информации

Электронная вычислительная машина обрабатывает информацию точно так же, как это делает человек. Для иллюстрации этого рассмотрим обработку информации человеком (рис. 1.4). При обработке информации человек использует свою память. Рассмотрим следующий простой пример. Пусть вам сообщили следующую информацию: числа 2, 5 и 3 являются исходными данными. Над ними следует выполнить следующие действия: умножить первое число на третье и запомнить результат. Затем следует сообщить результат.

Чтобы выполнить эту работу, необходимо воспользоваться своей памятью. В процессе ввода информации вы должны зафиксировать в своей памяти числа 2, 5 и 3. Результат умножения также нужно запомнить.

Данные (числа 2, 5 и 3) и *программа* (перемножить первое и третье число) вводятся с помощью органов слуха. В процессе ввода информации звуковые колебания преобразовываются в электрические импульсы¹, которые через нервную систему поступают в мозг и фикси-

¹ Правильнее было бы говорить о синаптических сигналах. (Прим. ред.)

руются в памяти. Затем мозг под управлением программы выполняет обработку информации. Результат, который хранится в памяти, выводится с помощью органов речи, являющихся устройством вывода. Устройство вывода данных производит слово «шесть».

Описанная здесь программа настолько проста, что ее можно выполнить мгновенно. В других случаях необходимо располагать более детальной информацией, чтобы выполнить поставленную задачу. Например, может потребоваться обращение к карточному файлу (картотеке) и выборка из него данных. В этом случае, прежде чем продолжить процесс вычислений, потребуется ввести эти данные в память.

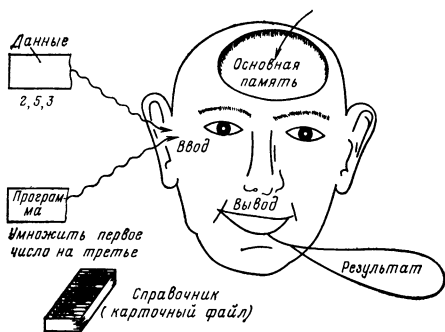


Рис. 1.4.

1.4. Структурная схема ЭВМ

Информация. Когда употребляются слова «информация» или «данные», то имеются в виду сведения, которые вводятся в ЭВМ. Данные могут быть представлены с помощью букв, знаков препинания, цифр, электрических сигналов, связанных с некоторым действием, и т. п. Эта информация помещается в *основную память*.

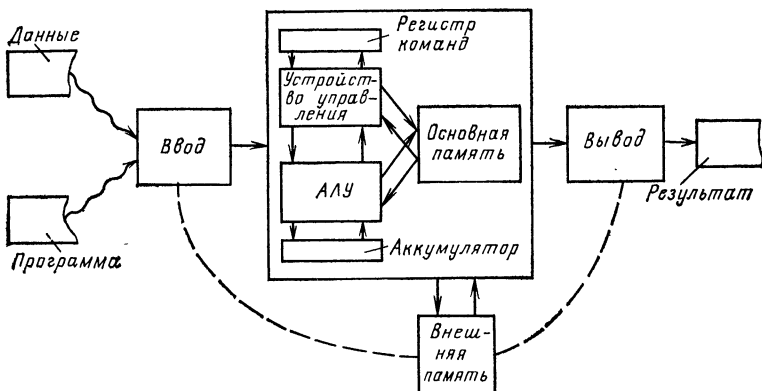


Рис. 1.5.

Программа состоит из совокупности команд. Команды, которые могут выполняться на некоторой ЭВМ, составляют ее *систему команд* (см. в приложении систему команд МП 8080). Программа, которая

должна быть выполнена, так же как и обрабатываемые данные, размещается в основной памяти.

Ввод. Данные и программа преобразуются в совокупность электрических сигналов с помощью устройства ввода. Электрические сигналы воздействуют на некоторые ячейки памяти так, что их состояние соответствует введенным данным и программе. Устройства ввода подробно рассмотрены в гл. 7; в гл. 20 приводятся некоторые дополнительные данные.

Устройство управления контролирует все взаимодействия, возникающие между различными частями ЭВМ. Для временного хранения данных устройство управления может использовать набор регистров. Один из этих регистров называется *регистром команд* и служит для размещения кода выполняемой команды, выбираемой из основной памяти. Команда, находящаяся в данный момент в регистре команд, сообщает устройству управления следующую информацию:

- а) *адрес ячейки памяти*, в которой находятся данные;
- б) *место*, в которое эти данные должны быть перемещены;
- в) *действие*, которое должно быть выполнено.

Таким образом, устройство управления с помощью программы управляет потоками информации в ЭВМ. После выполнения команды *счетчик команд*, имеющийся в устройстве управления, выдает адрес. Этот адрес обеспечивает выборку из основной памяти (и исполнение) следующей команды.

Арифметическо-логическое устройство (АЛУ) можно рассматривать как устройство, помогающее устройству управления выполнять команды. Оно может выполнять *арифметические* операции (например, сложение и вычитание). Кроме того, АЛУ способно выполнять *логические* операции, т. е. обрабатывать коды чисел с помощью логических операций И, ИЛИ, сложение по модулю два. В распоряжении АЛУ имеется также вспомогательная память в виде накапливающего регистра, который принято называть *аккумулятором*.

Когда команда, находящаяся в данный момент в регистре команд устройства управления, требует выполнения некоторой арифметической или логической операции над данными, эти данные извлекаются из основной памяти и посылаются в АЛУ. Затем АЛУ в соответствии с указанием устройства управления обрабатывает эти данные. После завершения операции результат (новые данные) запоминается в аккумуляторе.

Центральный процессор. В больших ЭВМ, которые используются в системах обработки данных и для научных расчетов, к центральному процессору принято относить устройство управления, АЛУ и память.

В микро-ЭВМ центральный процессор представляет собой сочетание устройства управления, АЛУ и связанных с ними регистров, выполняющих роль памяти. Такое толкование термина «центральный процессор» объясняется тем, что в микро-ЭВМ устройство управления и АЛУ реализуются в одном корпусе и физически отделены от основной памяти.

Поскольку в данной книге рассматриваются вопросы использования микро-ЭВМ, в термин «центральный процессор» будем вкладывать общепринятый смысл, т. е. будем считать, что центральный процессор — это совокупность устройств управления (УУ), АЛУ и связанных с ними регистров.

Устройства вывода данных представляют результат в удобной для визуального восприятия форме. По команде ЭВМ или оператора электрические сигналы, представляющие собой результат выполнения

операции, выводятся из микро-ЭВМ с помощью печатающего устройства или на экран дисплея. Устройства вывода более подробно описаны в гл. 20.

Выводы

1. Информация или данные могут быть представлены в виде слов, чисел и т. п., вводимых в ЭВМ для обработки.

2. Программа представляет собой совокупность команд, которые должны быть выполнены для получения требуемого результата.

3. Набор команд, которые может выполнять данная ЭВМ, образует систему команд этой ЭВМ.

4. Устройство управления ЭВМ обеспечивает взаимодействие различных частей машины. Устройство управления имеет в своем распоряжении несколько регистров, в том числе регистр команд.

5. Арифметические и логические операции выполняются в АЛУ. В составе АЛУ имеется свой регистр. Этот регистр называется аккумулятором.

6. Применительно к микро-ЭВМ термин «центральный процессор» относится к устройству, которое включает в себя УУ, АЛУ и связанные с ними регистры.

В больших ЭВМ в состав центрального процессора включается также основная память.

1.5. Память

Чтобы иметь возможность выполнять обработку данных по заданной программе, ЭВМ должна иметь память, в которой хранятся команды и обрабатываемые данные. Память вычислительной системы состоит из следующих частей (рис. 1.6),

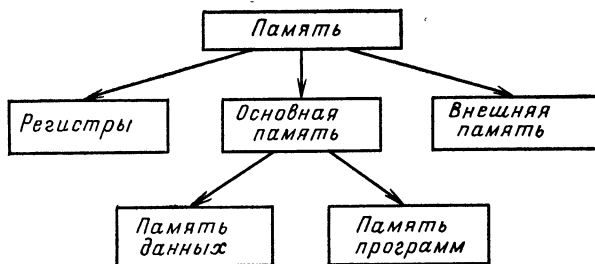


Рис. 1.6.

Основная память, в которой хранится следующая информация:

- 1) совокупность команд, образующих программу;
- 2) обрабатываемые данные.

Та часть основной памяти, в которой хранятся команды, называется *памятью программ*. Если предполагается использовать ЭВМ только для решения одной задачи, программа может постоянно храниться в основной памяти. Такая ЭВМ называется специализированной. Подобная ситуация является очень распространенной в микро-

ЭВМ. В соответствии с требованиями пользователя изготовитель микро-ЭВМ выпускает постоянную память программ.

Если ЭВМ является многоцелевой, то должна быть обеспечена возможность смены программ. Типичным примером является исполь-



Рис. 1.7.

зование ЭВМ для начисления зарплаты служащим или для управления запасами в различных организациях. Чтобы выполнить некоторую программу, ее следует ввести в основную память.

Примечание. Постоянные ЗУ (ПЗУ) могут использоваться и в многоцелевых ЭВМ, если программы не очень многочисленны и не требуют слишком большого объема памяти. В этом случае различные программы хранятся в ПЗУ и вызываются по мере необходимости.

Та часть памяти, которая используется для временного хранения данных, называется *оперативным запоминающим устройством* (ОЗУ). В ОЗУ временно хранятся результаты выполнения операций до вывода их на внешнее устройство.

Внешняя память. Если требуется организовать хранение данных и программ с целью их последующего использования, то применяются *внешние запоминающие устройства* (ВЗУ). Чаще всего — это накопители на магнитной ленте или на магнитном диске. На рис. 1.7 показан внешний вид накопителя на магнитной ленте (НМЛ).

В микро-ЭВМ обычно используются НМЛ кассетного типа. Используются кассеты подобны тем, которые применяются в современных бытовых кассетных магнитофонах.

В качестве накопителя на магнитных дисках (НМД) используется накопитель на гибком диске. Носитель информации в этом случае представляет собой гибкий диск с магнитным покрытием, помещаемый в специальную кассету (рис. 1.8). Сам диск очень напоминает грампластинку с той лишь разницей, что информацию на нем можно стирать и повторно записывать, как на магнитной ленте.

Регистры представляют собой вспомогательную память для временного хранения информации, распределенную по всей системе. Регистры

стры имеются не только в АЛУ и УУ, но и в устройствах ввода-вывода.

Примечание. Термин ЭВМ иногда используют для того, чтобы обозначать не всю систему в целом, а только ее центральную

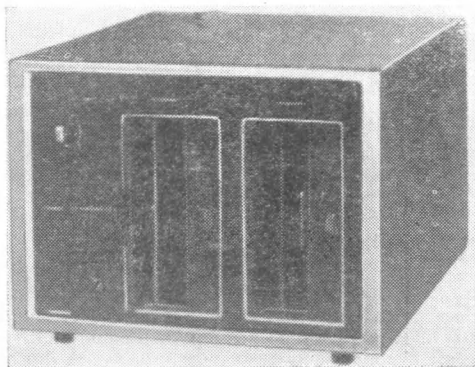


Рис. 1.8.

часть, за исключением периферийных устройств. Вся система в целом (включая периферийные устройства) называется вычислительной системой.

В данной книге используем термин ЭВМ для обозначения всей вычислительной системы.

Выводы

1. Программа, описывающая вычислительный процесс, хранится в основной памяти и может быть запущена, когда это необходимо.

2. Обрабатываемые данные перед выполнением операции должны быть помещены в основную память.

3. Память ЭВМ подразделяется на основную, внешнюю и регистры.

4. Основную память принято делить на две части: память данных, в которой хранятся обрабатываемые данные, и память программ, в которой находится совокупность команд.

5. Специализированная ЭВМ ориентирована на решение одной единственной задачи.

Универсальная ЭВМ предназначена для многоцелевого использования.

6. Внешние запоминающие устройства используются для длительного хранения программ и данных. Типичными ВЗУ являются НМЛ, НМД, кассетные НМЛ и накопители на гибких дисках.

1.6. Организация основной памяти

Команды и данные, расположенные в основной памяти, должны быть легко доступны. Память состоит из блоков одинакового размера (рис. 1.9). Эти блоки называются *словами* или *ячейками памяти*.

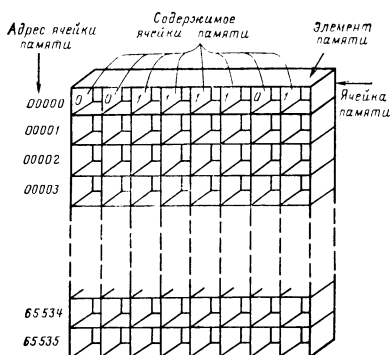


Рис. 1.9.

Каждое слово (ячейка) памяти имеет свой *адрес*. В большинстве микро-ЭВМ максимальное значение адреса составляет $2^{16} = 65\,536$. Этим числом ограничивается размер основной памяти. Поскольку большие значения адресов не допустимы, большее число ячеек памяти использовать нельзя.

1.7. Устройства ввода-вывода

Устройство ввода с перфоленты и ленточный перфоратор. Перфолента — очень распространенный носитель данных для ввода информации в микро-ЭВМ. Она представляет собой длинную и узкую полосу бумаги, на которой информация кодируется пробивками круг-

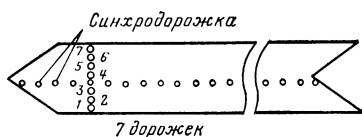


Рис. 1.10.

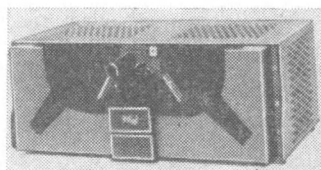


Рис. 1.11.

лой или квадратной формы. Один символ кодируется на перфоленте набором пробивок, расположенных в одной строке в поперечном направлении ленты. В зависимости от используемого способа кодирования число пробивок равно 5—8. На рис. 1.10 приведен пример перфоленты с 7-позиционным кодом. Используя 7-позиционный код, можно представить в общей сложности $2^7 = 128$ различных символов. Для обеспечения возможности протяжки ленты на нее в середине нанесены специальные отверстия меньшего диаметра (синхродорожка).

На рис. 1.11 показано устройство ввода с перфоленты и вывода на перфоленту (ленточный перфоратор). При вводе данных в ЭВМ

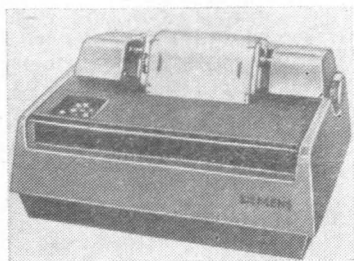


Рис. 1.12.

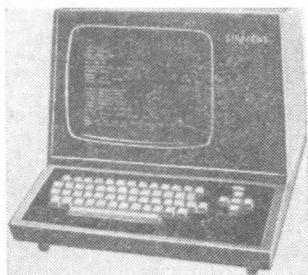


Рис. 1.13.

это устройство преобразует комбинации пробивок в электрические сигналы. При выводе информации из ЭВМ оно преобразует электрические сигналы в пробивку на бумажной ленте.

Построчно печатающее устройство (рис. 1.12) наиболее часто используется как средство вывода информации. Бумажный носитель в таком устройстве обычно сложен в виде «гармошки», образующей непрерывную полосу бумаги.

Дисплей. В строгом смысле этого слова дисплей представляет собой телевизионный экран, на который информация выводится в виде текста, цифр или графиков. Чаще всего дисплей дополняется клавиатурой, которая используется как средство ввода. Такая совместная конфигурация часто также называется дисплеем (рис. 1.13). С помощью клавиатуры можно запросить информацию из ЭВМ и увидеть ее на экране.

Телетайп (рис. 1.14) обеспечивает как ввод, так и вывод данных. Телетайп представляет собой пишущую машинку, которая используется программистом для ввода данных, и ЭВМ для вывода данных. Кроме того, в состав телетайпа входит устройство ввода с перфолененты и ленточный перфоратор. Телетайп — самое распространенное устройство ввода-вывода в микро-ЭВМ.

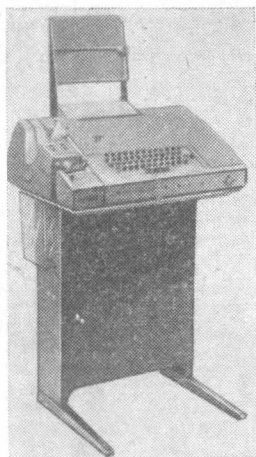


Рис. 1.14.

Выводы

1. Память ЭВМ должна быть четко организована. Для этой цели она разделена на ячейки (слова) равной длины. Обычно длина ячеек памяти в микро-ЭВМ равна 8 бит (1 байт).

2. Каждая ячейка памяти имеет адрес и содержимое. Адрес ячейки неизменен. Содержимое ячейки можно изменить.

3. С помощью устройства ввода неэлектрические сигналы преобразуются в электрические, которые могут быть обработаны в ЭВМ. Устройство вывода преобразует электрические сигналы, поступающие от ЭВМ, в визуальную, графическую или иную форму представления информации.

4. Наиболее часто в составе микро-ЭВМ используются такие устройства ввода-вывода, как телетайп, устройство ввода с перфоленты, ленточный перфоратор и построчно печатающее устройство.

1.8. Блок-схема алгоритма

Хорошо запрограммированная ЭВМ может выполнять вычисления очень быстро. Применять ЭВМ наиболее выгодно в тех случаях, когда одна и та же процедура может быть многократно использована для обработки различных данных. При этом процедура вычислений должна быть сообщена ЭВМ только 1 раз.

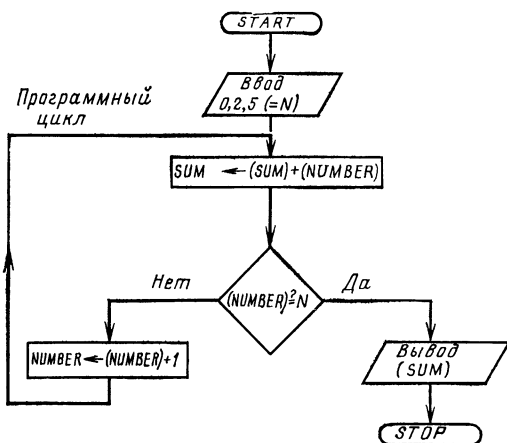


Рис. 1.15.

Процедура, которую может выполнять ЭВМ, называется *программой*. Отдельные шаги этой процедуры называются *командами*. Чтобы запрограммировать ЭВМ для выполнения определенной задачи, необходимо формально описать последовательность выполняемых операций, которые приводят к ее решению. Такое формальное описание называется *блок-схемой алгоритма*, или коротко *блок-схемой* (рис. 1.15).

При разработке блок-схемы алгоритма следует учитывать конкретные особенности используемой ЭВМ. Форма блок-схемы зависит от субъективных особенностей разрабатывающего ее человека. Процесс разработки блок-схемы требует не только знаний и опыта, но и определенных творческих способностей. Программирование ЭВМ — это одновременно наука и искусство. Люди обладают различными творческими способностями и различной подготовкой.

На простом примере познакомим читателя с процессом разработки блок-схем. Более подробно эти вопросы обсуждаются в гл. 14.

Задача. Найти сумму ряда последовательных положительных целых чисел от P до N . Числа P и N могут быть любыми. Возьмем для примера $P = 2$ и $N = 5$.

Примечание. Следует иметь в виду, что:

а) за одну команду ЭВМ может выполнить сложение только двух чисел;

б) число данных, которое вводится в машину, должно быть минимальным.

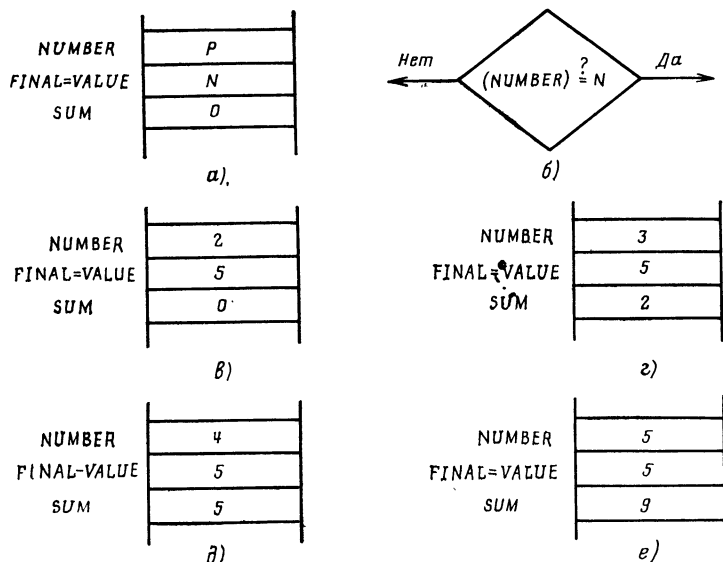


Рис. 1.16.

В рассматриваемом случае число вводимых данных будет минимальным, если ввести только начальное значение P и конечное значение N . Используя начальное значение, ЭВМ может вычислить все промежуточные значения членов ряда. Именно так и поступим. Начальное значение P загружается в ячейку памяти с *символическим адресом* *NUMBER* (рис. 1.16, а).

Примечание. Из последующих примеров разработки программ увидим, что использование символических адресов — это распространенный прием. Символический адрес задают в виде имени. Впоследствии этот адрес преобразовывается в действительный адрес памяти ЭВМ. Вычисление действительного адреса памяти осуществляется с помощью транслятора (программы-ассемблера). Этот вопрос подробно рассмотрен в гл. 18, а некоторые сведения об этом приведены в гл. 3.

Когда число P будет обработано, адресу *NUMBER* будет присвоено следующее значение. Это действие символически будем обозначать как

$$\text{NUMBER} \leftarrow (\text{NUMBER}) + 1.$$

Такую символическую запись можно расшифровать следующим образом. В ячейку памяти с символическим адресом **NUMBER** посылается число, равное сумме прежнего значения содержимого памяти с адресом **NUMBER** и 1. *Если необходимо указать содержимое ячейки памяти, то ее адрес заключают в скобки.* Содержимое ячейки **NUMBER** прекращают увеличивать на 1 (инкрементировать) после того, как оно станет равным **N**. Для этого необходимо постоянно сравнивать содержимое ячейки с адресом **NUMBER** с содержимым ячейки памяти, в которую помещено значение **N**. В рассматриваемом случае значение **N** было размещено в ячейке памяти с символическим адресом **FINAL = VALUE**. Операция сравнения содержимого ячейки памяти **NUMBER** с величиной **N** показана на рис. 1.16, б. Если содержимое ячейки **NUMBER** равно значению **N**, то программа продолжается в направлении **ДА**. Если содержимое не равно **N**, то программа продолжается в направлении **НЕТ**.

Поскольку ЭВМ за одну операцию может сложить только два числа, поступают следующим образом. Помечают ячейку, в которой накапливается сумма, символическим адресом **SUM**. Вначале посылают в эту ячейку значение 0. Затем на каждом шаге суммирования увеличивают содержимое ячейки **SUM** на соответствующее значение. Сначала прибавляют к содержимому ячейки **SUM** первое число. После этого содержимое ячейки **SUM** будет равно $0 + P = P$. Затем прибавляют к содержимому ячейки **SUM** второе число. Содержимое ее будет равно $P + (P + 1) = 2P + 1$. К новому содержимому прибавляется третье число. Содержимое становится равным $2P + 1 + (P + 2) = 3P + 3$ и т. д. Этот процесс символически изображается следующим образом:

$$\text{SUM} \leftarrow (\text{SUM}) + (\text{NUMBER}).$$

Запись расшифровывается так. В ячейку с символическим адресом **SUM** посылается результат сложения прежнего содержимого ячейки **SUM** и содержимого слова памяти с символическим адресом **NUMBER**. Следует иметь в виду, что после каждой операции суммирования содержимое ячейки **NUMBER** увеличивается на 1.

Решение. Метод решения задачи представлен на блок-схеме, показанной на рис. 1.15. Оператор **START** указывает начальную точку программы. Затем вводятся исходные данные: в рассматриваемом случае ячейки памяти с символическими адресами **NUMBER**, **FINAL = VALUE** и **SUM** заполняются значениями 2, 5 и 0 соответственно (рис. 1.16, в). Затем начинается процесс вычислений:

$$\text{SUM} \leftarrow (\text{SUM}) + (\text{NUMBER}).$$

Содержимое адреса **SUM** становится равным $0 + 2 = 2$. Затем необходимо сравнить содержимое адреса **NUMBER** с числом 5.

Содержимое адреса **NUMBER** равно 2. Поэтому результат сравнения соответствует ситуации *Нет*. Этот процесс называется программным циклом (цикл выполняется до тех пор, пока содержимое адреса **NUMBER** не станет равным 5.)

Вслед за этим выполняется операция

$$\text{NUMBER} \leftarrow (\text{NUMBER}) + 1.$$

В результате этой операции содержимое ячейки **NUMBER** станет равным $2 + 1 = 3$. Содержимое ячеек памяти показано на рис. 1.16, г.

Далее повторяется вычисление $SUM \leftarrow (SUM) + (NUMBER)$. Содержимое ячейки SUM становится равным $2 + 3 = 5$. Содержимое ячейки $NUMBER$ сравнивается с числом 5. Результат сравнения приводит к ситуации **НЕТ**. Поэтому следует вновь повторить операцию $NUMBER \leftarrow (NUMBER) + 1$.

Содержимое адреса $NUMBER$ становится равным $3 + 1 = 4$. Содержимое ячеек памяти показано на рис. 1.16, *д*. Затем повторяется вычисление $SUM \leftarrow (SUM) + (NUMBER)$.

Содержимое адреса SUM станет равным $5 + 4 = 9$. Далее выполняется сравнение. Поскольку содержимое адреса $NUMBER$ все еще меньше 5, осуществляется операция $NUMBER \leftarrow (NUMBER) + 1$. Содержимое адреса $NUMBER$ станет равным $4 + 1 = 5$. Содержимое остальных ячеек показано на рис. 1.16, *е*.

Затем повторяется процедура вычисления $SUM \leftarrow (SUM) + (NUMBER)$. В результате содержимое адреса SUM становится равным $9 + 5 = 14$. Затем снова выполняется сравнение. Поскольку содержимое адреса $NUMBER$ стало равно N , подсчитанное значение суммы (число 14) выводится на печать и программа останавливается.

1.9. Языки программирования

Процедура, которую должна выполнять вычислительная машина, называется *программой*. Отдельные шаги программы называются *командами*. Для написания программ могут использоваться *машинно-ориентированные* или *проблемно-ориентированные* языки. Вместо термина проблемно-ориентированный язык часто используют термин *язык высокого уровня*.

Языки программирования

Проблемно-ориентированные:

БЕЙСИК
ФОРТРАН
ПЛ/М
КОБОЛ

Машинно-ориентированные:

АСЕМБЛЕР

В операторах машинно-ориентированного языка непосредственно учитываются особенности соответствующей ЭВМ.

Недостатком написания программ на машинно-ориентированном языке является то, что программист должен хорошо знать особенности организации соответствующей ЭВМ.

Преимуществом использования машинно-ориентированного языка является то, что получаются более короткие программы и для их хранения требуется меньший объем памяти.

Разработка программ для микро-ЭВМ с использованием машинно-ориентированного языка имеет определенные экономические преимущества. Машинно-ориентированный язык микро-ЭВМ называется *языком ассемблера*. Поскольку микро-ЭВМ, производимые различными изготовителями, не похожи друг на друга, отличаются и языки ассемблера. Однако эти отличия невелики.

При написании программ на проблемно-ориентированном языке программист должен хорошо владеть возможностями этого языка. В этом случае программа пишется, исходя из существа задачи. Проблемно-ориентированными языками программирования для микро-

ЭВМ являются БЕЙСИК и ПЛ/М. Аббревиатура ПЛ/М (PL/M) получена от английского названия Programming Language for Microcomputures (язык программирования для микро-ЭВМ). КОБОЛ используется для разработки программ решения учетных и экономических задач. ФОРТРАН используется для программирования научных прикладных задач.

Выводы

1. Наиболее часто используемым языком программирования микро-ЭВМ является язык ассемблера. При помощи ассемблера (транслирующей программы) программа на языке ассемблера преобразуется в программу на машинном языке.

2. Если программа составляется для решения некоторой задачи, без учета конкретных особенностей ЭВМ, следует использовать для этой цели языки программирования высокого уровня.

1.10. Язык ассемблера

Электронная вычислительная машина организована таким образом, что, когда некоторая команда выбирается из основной памяти и поступает в УУ, автоматически выполняется ряд операций.

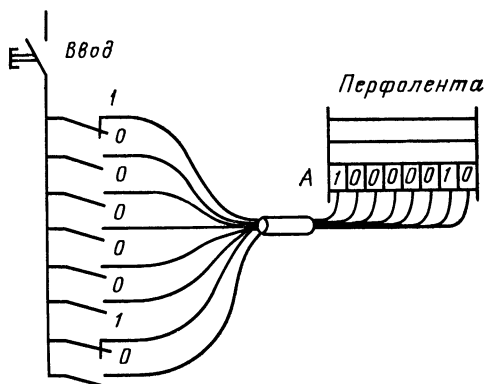


Рис. 1.17.

Система команд ЭВМ представляет собой совокупность команд, которые могут выполняться на данной машине. Приведем пример типичной команды микро-ЭВМ:

Команда	Код команды
ADD D	10000 010

Данная команда означает, что содержимое регистра D в АЛУ должно быть прибавлено к содержимому аккумулятора, а сумма помещена в аккумулятор (назначение команд будет пояснено в гл. II).

Код операции можно задать различными способами, один из которых показан на рис. 1.17. Команда вводится с помощью клавишных (тумблерных) переключателей (один переключатель соответствует

одному двоичному разряду). Нажатие клавиши ввода приводит к тому, что биты, соответствующие замкнутым контактам, устанавливаются в состояние 1.

Последовательность команд может быть таким образом введена в последовательные ячейки памяти. Команды, представленные в двоичных кодах, образуют программу на машинном языке. Если программа представлена в машинных кодах и может быть введена в ЭВМ, она называется объектной программой (рис. 1.18).

Если необходимо решить некоторую задачу, например создать систему автоматического управления светофором дорожного движения, то в конечном счете необходимо составить объектную программу, поскольку это единственная форма программы, доступная пониманию ЭВМ. Это не значит, что следует сразу писать программу как объектную. У такой программы есть два недостатка:

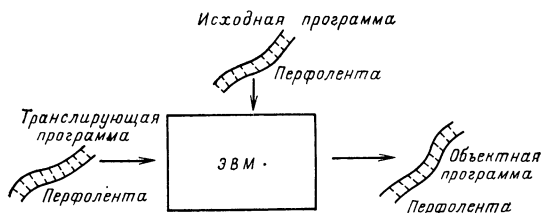


Рис. 1.18.

а) из-за большой длины команд легко перепутать единицы и нули и совершить ошибку;

б) объектный код очень трудно воспринимается человеком.

По этой причине человек пишет программу с использованием символического языка, в котором используются аббревиатуры для обозначения выполняемых операций. Буквенные коды операций называются mnemonic кодами. Команда сложения может, например, иметь mnemonic код (*мнемонику*) ADD (от addition — сложение). Команда вычитания может иметь мнемонику SUB (от subtraction — вычитание). Операция пересылки может быть обозначена MOVE (переслать по-английски). Программа, написанная с использованием подобных mnemonic обозначений, называется *исходной программой*.

1.11. Программы-трансляторы

После того, как программа написана на языке ассемблера, БЕЙСИК или ПЛ/М, она должна быть преобразована в объектную программу.

Команды исходной программы перфорируются с помощью телепайпа, в составе которого имеется ленточный перфоратор. При вводе перфоленты пробивки на ней преобразуются в электрические сигналы, воспринимаемые ЭВМ.

Программа-транслятор, как и любая другая программа, должна быть нанесена на перфоленту и введена в ЭВМ. С помощью программы-транслятора ЭВМ преобразует исходную программу в объектную.

Объектная программа может быть размещена на перфоленте или магнитной ленте с тем, чтобы использоваться впоследствии.

Программа-транслятор для программ, написанных на языке ассемблера, называется *ассемблером*.

Программа-транслятор для программ, написанных на языке высокого уровня, называется *компилятором*.

Трансляторы подробно рассмотрены в гл. 18

1.12. Шестнадцатиричная система счисления

В ЭВМ адреса и содержимое ячеек памяти представлены в двоичной системе счисления. При работе с двоичной системой счисления очень легко ошибиться. По этой причине в языке пользователя применяется шестнадцатиричная система счисления. В шестнадцатиричной системе счисления группы из 4 бит представляются одной шестнадцатиричной цифрой. Шестнадцатиричная система счисления использует 16 цифр: от 0 до 9 и от А до F.

Выводы

1. Команды объектной программы записаны двоичными кодами.
2. Программа, представленная в форме, отличной от объектной, называется исходной программой.
3. Программа-транслятор для исходных программ, написанных на языке ассемблера, называется ассемблером.
4. Программа-транслятор для программ, написанных на языке высокого уровня, называется компилятором.
5. При описании информации, хранимой в ЭВМ, используется шестнадцатиричная система счисления.

Глава вторая ЧТО ТАКОЕ МИКРО-ЭВМ

2.1. Введение

В предыдущей главе было отмечено, что микро-ЭВМ — это ЭВМ с центральным процессором на одной интегральной микросхеме (иногда на двух); при этом ЦП называется микропроцессором. В этой главе рассмотрим организацию микро-ЭВМ на уровне структурной схемы.

Сначала немного об истории электронно-вычислительной техники и происхождении микро-ЭВМ.

В 1943 г. Экерт, Макли и Голдстейн начали разработку первой ламповой цифровой ЭВМ ENIAC в Пенсильванском университете. Эта ЭВМ (электронный числовой интегратор и калькулятор) содержала 18 тыс. ламп и потребляла 150 кВт. Она была специально разработана для баллистических вычислений в артиллерии США и работала примерно в 1000 раз быстрее немногих имеющихся тогда компьютеров релейного типа, созданных в 1935 г.

В 1946 г. известный математик Джон фон-Нейман выдвинул следующую идею: если с командами обращаться так же, как с информацией, то программа может модифицироваться сама в ходе ее исполне-

ния. С помощью команд перехода или ветвления выполнение или невыполнение определенных частей программы будет зависеть от результата определенных операций.

Экерт, Макли и фон-Нейман разработали первую так называемую ЭВМ с хранимой программой UNIVAC = 1, основанную на этой идее. В этой ЭВМ информация и программа хранились в памяти и включались в работу по мере необходимости.

В 1958 г. с появлением транзисторов габариты ЭВМ существенно уменьшились.

В 1965 г. интегральные микросхемы заменили отдельные компоненты и ЭВМ стали еще меньше, а быстродействие их возросло.

В 1959 г. разработчики фирмы Datapoint (США) сделали важный шаг вперед. Они разработали очень простой блок управления и арифметический блок и обратились к фирмам Texas Instruments и Intel для реализации своей разработки.

Фирма Intel добилась успехов, но при этом выяснилось, что первый микропроцессор работал в 10 раз медленнее, чем ожидала Datapoint, которая в связи с этим отказалась от дальнейшего сотрудничества. Однако фирма Intel продолжала придерживаться прототипа, средства на разработку которого уже были израсходованы. Она могла бы не использовать этот прототип, но рискнула начать производство ЭВМ на его основе. Таким образом, хорошо известный МП Intel 8008 стал первым микропроцессором на мировом рынке.

Пример состава и размещения отдельных элементов микро-ЭВМ дан на рис. 2.1, где 1 — ПЗУ ($3 \times D464$) для хранения программ-монитора емкостью 3/4 Кбайт); 2 — позиция для корпуса ППЗУ или РППЗУ для хранения программ пользователя (емкость 1/4 Кбайт); 3 — дисплей (восемь 7-сегментных светодиодных индикаторов для отображения адресов и содержимого ячеек памяти); 4 — клавиатура (16 клавиш шестнадцатиричного кода и 9 функциональных клавиш); 5 — модуль ввода-вывода 8255 (3 порта ввода-вывода); 6 — свободная зона (применяется для монтажа специальных схем пользователя, например, для интерфейса с кассетным накопителем); 7 — блок управления и буферный усилитель 8228; 8 — микропроцессор (ЦП) 8080; 9 — переключатель автоматического или пошагового исполнения программы; 10 — переключатель защиты данных (в положении «защита» ОЗУ переключается на батарейный источник электропитания 3 В); 11 — блок синхронизации 8224 и кварцевый осциллятор с частотой 18 432 МГц (после деления частоты на 9 в ЦП сигналы синхронизации поступают с частотой 2048 МГц); 12 — ОЗУ $4 \times D5101$ (может содержать рабочую программу и данные, емкость 1/2 Кбайт, но может быть расширена до 1 Кбайт).

2.2. Организация микро-ЭВМ

На рис. 2.2 представлена структурная схема микро-ЭВМ. Как видно из рис. 2.2, для ввода данных в ЭВМ могут быть использованы обычные периферийные устройства: гибкий диск, телетайп и устройство считывания с перфоленты. Данные от процесса или объекта управления могут быть введены непосредственно. Если данные имеют аналоговую форму, например, для управления температурой на промышленном предприятии, они сначала должны быть преобразованы в цифровую форму, так как ЭВМ может работать только по двоичной системе. Это преобразование осуществляется с помощью аналого-цифрового

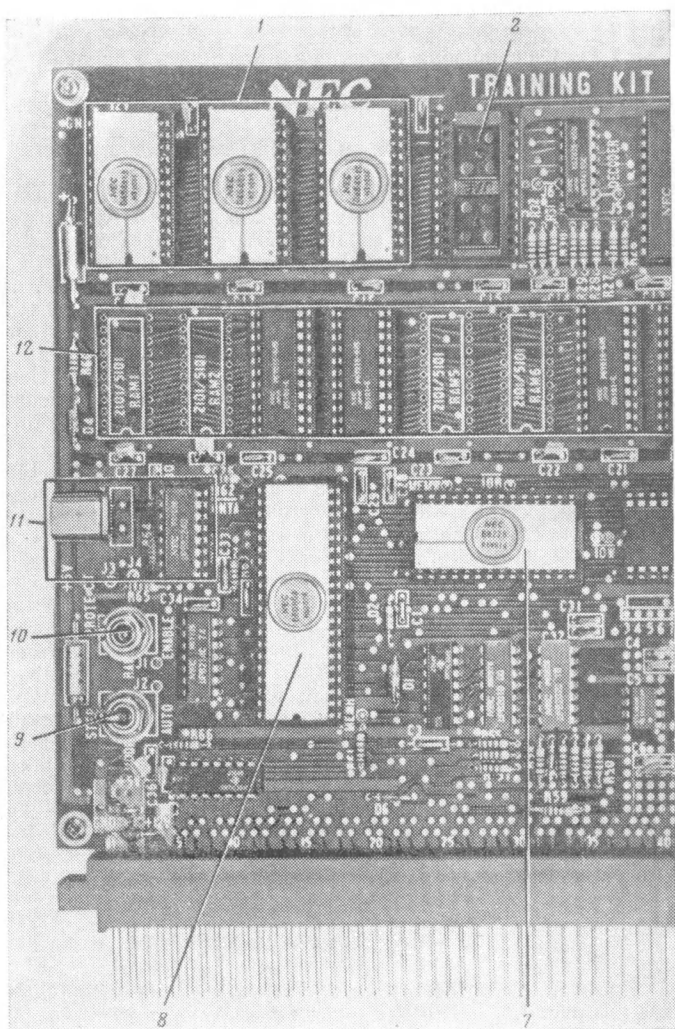
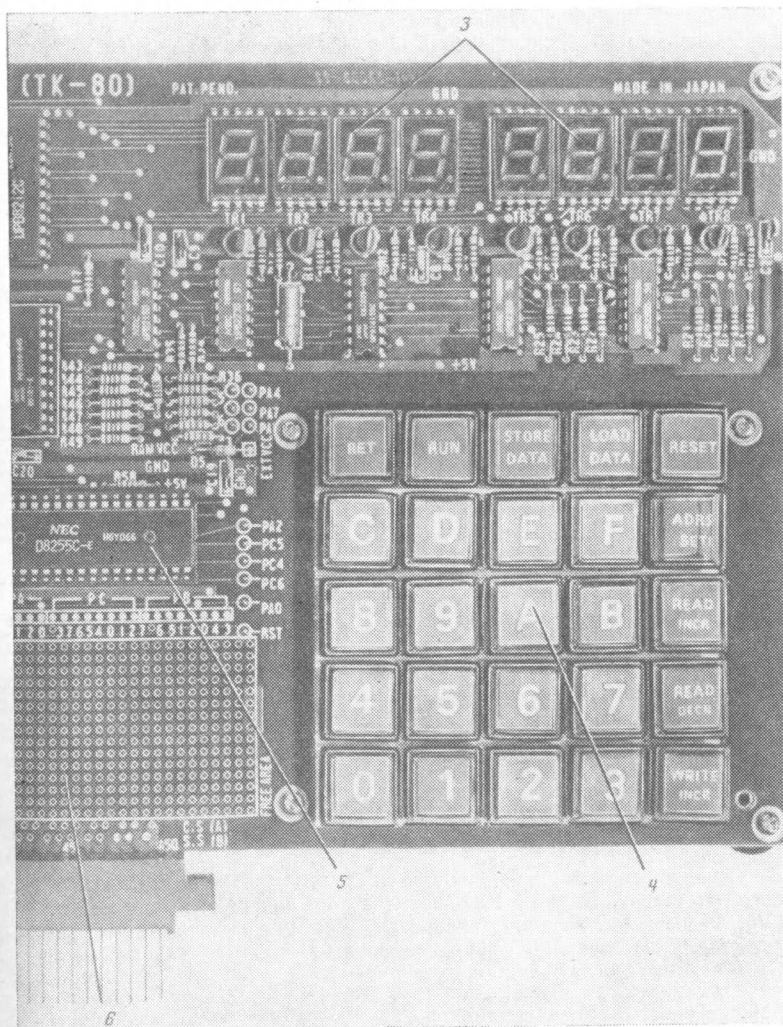


Рис. 2.1.



преобразователя (АЦП). Аналогично данные для управления процессом, полученные на выходе ЭВМ, могут быть преобразованы в аналоговую форму с помощью цифро-аналогового преобразователя (ЦАП).

Адреса, данные и сигналы управления передаются по шинам.

Данные, поступающие с *устройства ввода*, передаются по *шине данных* в виде 8-разрядных параллельных или последовательных кодовых сигналов через порт ввода. *Селектор адреса* определяет порт ввода, который передает данные на шину данных в некоторый момент времени.

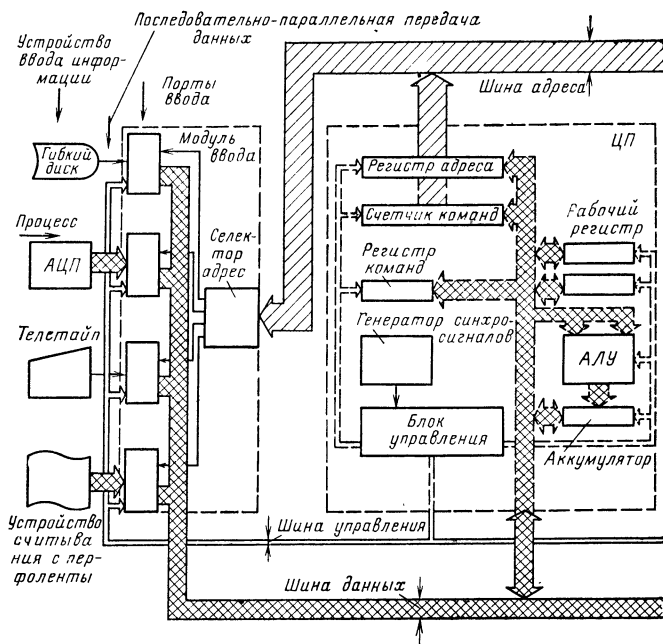


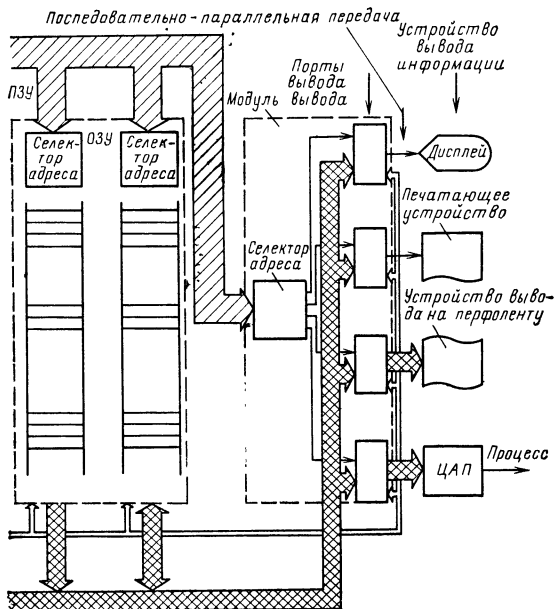
Рис. 2.

Основная память состоит из ПЗУ и ОЗУ. Содержимое ПЗУ не может быть стерто. Поэтому ПЗУ используется как память программы, которую изготовитель заранее запрограммировал в соответствии с требованиями пользователя. Для различных программ используют различные ПЗУ или их части. Постоянное запоминающее устройство можно сравнить со складом, с которого можно только получать какие-то вещи (предметы). Различные виды ПЗУ более подробно рассмотрены в гл. 3.

Памятью данных в микро-ЭВМ является ОЗУ. Его можно сравнить со складом, на который можно сдавать вещи и получать. Информация, хранящаяся в ОЗУ, стирается, когда прерывается подача напряжения питания. Это можно предотвратить, установив резервные батареи, подсоединенные параллельно к источнику питающего напряжения.

В дальнейшем, используя термины ПЗУ и ОЗУ, необходимо помнить, что ОЗУ (RAM) — это память данных, ПЗУ (ROM) — память программы.

Данные, поступающие в ОЗУ, обрабатываются в ЦП в соответствии с программой, хранящейся в ПЗУ. Результаты операций в ЦП хранятся в аккумуляторе или ОЗУ. Они могут быть выведены по команде через один из *портов вывода на устройства вывода*, подсоединенные к этому порту. Требуемый порт вывода выбирается через схему селекции адреса.



Выводы

1. Различные части микро-ЭВМ соединены друг с другом системой шин.

2. Шины состоят из ряда линий данных, адресных и управления.

3. В дополнение к обычным периферийным устройствам данные от процесса могут быть введены в микро-ЭВМ непосредственно. Данные должны быть преобразованы в цифровую форму, так как ЭВМ может оперировать только с сигналами 0 и 1, а данные должны быть представлены в двоичной системе счисления.

4. Памятью данных в микро-ЭВМ является ОЗУ, а памятью программы — ПЗУ.

2.3. Система шин микро-ЭВМ

Для соединения различных блоков ЭВМ используется так называемая «*шинная структура*». Шина — это многожильный кабель. В микро-ЭВМ (см. рис. 2.2) имеется три различные шины: адреса, данных и управления.

Шина адреса. Число линий в шине адреса определяется числом разрядов в адресе памяти. Большинство микро-ЭВМ имеют 16-разрядные адреса.

Шина данных. Число линий равно длине слова микро-ЭВМ, которое почти всегда 8-разрядное. Таким образом, шина данных имеет восемь линий.

Шина управления содержит линии управления, число которых зависит от типа микро-ЭВМ. Сигналы, включающие различные блоки в работу, передаются по шине управления.

Когда говорят о шинах в микро-ЭВМ, то имеют в виду *внутренние* шины. Микро-ЭВМ имеет разветвленную структуру внутренних шин, которая показана на рис. 2.2 пунктиром. Стрелками показано, принимает и(или) передает сигналы данный блок. Если сигналы и принимаются и передаются, то это соединение называется *двунаправленным*.

Примечание 1. Шинная структура позволяет прямо подключать к микро-ЭВМ новые блоки. Это имеет большое значение. Так, например, к системе, показанной на рис. 2.1, можно подсоединить четыре дополнительных блока ОЗУ и один блок ПЗУ.

Примечание 2. Шина данных является двунаправленной. Это, естественно, не относится к соединению между шиной данных и ПЗУ.

2.4. Система соединения блоков микро-ЭВМ

Информация может поступать на данную шину только от одного блока микро-ЭВМ. Например, информация может поступить на шину данных только от одного из следующих блоков: ЦП, ОЗУ, ПЗУ, од-

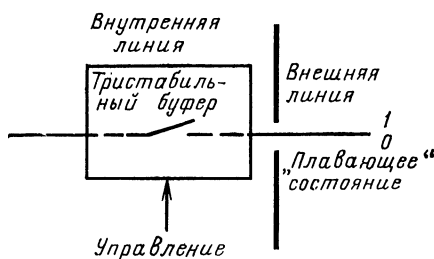


Рис. 2.3.

ного из портов ввода. Для достижения этого в каждую выходную линию шины каждого порта включена так называемая тристабильная буферная схема (рис. 2.3), которая аналогична выключателю, управляемому соответствующим сигналом.

Название тристабильная (tri-state) происходит от того, что для данного блока внешняя линия может принимать три разных состояния.

Когда под воздействием сигнала управления тристабильная буферная схема срабатывает, то соединения между внутренней и внешней линиями нет. В этом состоянии сопротивление между внутренней и внешней линиями высоко, и поэтому такое состояние называется высокоимпедансным. Его также часто называют плавающим.

Сигналы, которые переключают тристабильные буферные схемы в требуемое состояние, поступают от УУ. Таким образом, УУ выбирает блок ЭВМ, который передает данные в шину.

2.5. Устройство управления

Центральный процессор работает циклично, т. е. непрерывно выполняет одни и те же действия. Другими словами, ЦП непрерывно осуществляет выборку команды из памяти, выполняет операцию, указанную в команде, выбирает следующую команду и т. д.

Эта последовательность действий требует *синхронизации*. С этой целью в микро-ЭВМ встроен блок синхронизации, который совместно с УУ выдает необходимые сигналы для всех режимов работы микро-ЭВМ.

2.6. Счетчик команд

Последовательность команд, образующая программу, хранится в памяти программ. Центральный процессор выбирает эти команды в той же последовательности из памяти для определения операций, которые ему нужно выполнить. Это, однако, предполагает, что ЦП должен знать, где найти команды. Центральный процессор сохраняет адрес памяти программ на регистре, с помощью которого можно найти следующую команду. Этот регистр называется *счетчиком команд* (СК). При каждом выполнении команды ЦП инкрементирует (увеличивает на 1) содержимое счетчика команд. Таким образом, счетчик команд всегда содержит адрес следующей команды, которую необходимо выполнить. Поэтому программист должен разместить команды в последовательных ячейках памяти программ. Команды выполняются в возрастающей последовательности адресов: 00; 01; 02 и т. д.

Программист может, используя *команду перехода*, прервать эту последовательность и перейти к любой другой ячейке памяти вместо следующей. Команда перехода содержит адрес следующей команды, подлежащей выполнению. Таким образом, следующая команда может храниться в ячейке оперативной памяти до тех пор, пока ее адрес не будет задан в предыдущей команде. *Переходы по программе облегчают работу программиста*. Одним из важнейших преимуществ программных переходов является то, что программы, содержащие часто выполняемые фрагменты программы, являются не такими длинными, как если бы они были записаны полностью. Можно вернуться назад и повторить часть программы. Использование программных переходов сокращает требуемый объем памяти.

2.7. Выборка команды

Команда выбирается из памяти программы в два этапа (рис. 2.4).

На первом этапе ЦП передает адрес, заданный счетчиком команд, в память программ. Из памяти выбирается слово, расположенное

в ячейке памяти с этим адресом. На втором этапе из памяти программ передается *содержимое* адресуемой ячейки памяти в ЦП. Центральный процессор помещает это содержимое, являющееся командой, в так называемый *регистр команд*. Затем ЦП использует содержимое регистра команд для управления операциями, которые должны выполняться по данной команде. При этом инкрементируется содержимое счетчика команд.

Схема, в которой процессор преобразует код команды в специальные управляющие сигналы, называется *дешифратором команд*.

Одна часть команды, которая хранится в регистре команд, указывает, *какая операция* должна быть выполнена (например, сложение). Эта часть называется *кодом операции*. Другая часть команды указывает место, где находятся данные, участвующие в операции. Эта часть называется *операндом*.

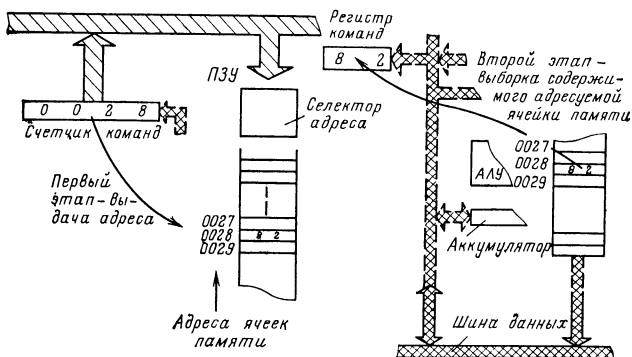


Рис. 2.4.

Команда ADD D 10000010 (см. гл. 1) означает: прибавить содержимое регистра D к содержимому аккумулятора. Здесь 10000 — код операции, который означает, что операнд должен быть прибавлен к содержимому аккумулятора; 010 — это адрес, который указывает, где находится операнд (в данном случае в регистре D, одном из регистров общего назначения ЦП). Это пример однобайтной команды, т. е. команды, которая занимает одну ячейку памяти в 8-разрядной микро-ЭВМ.

Команды могут иметь формат в 2—3 байта. При этом в 8-разрядной ЭВМ требуются 2—3 ячейки памяти для хранения таких команд. Такой формат команд необходим, когда, например, команда содержит адрес данных.

В микро-ЭВМ с ЦП, способным адресовать 65 536 ячеек памяти, адреса будут 16-разрядными, т. е. 2-байтными. В этом случае вся команда займет 3 байта (рис. 2.5). Первый байт является кодом операции и показывает, какая операция должна быть выполнена. Перед выполнением операции код операции поступает в регистр команд. Два других байта, которые указывают *адрес операнда* в памяти данных, поступают на *регистр адреса*.

Примечание. Во время выполнения 3-байтной команды содержимое счетчика команд инкрементируется трижды. Если счетчик

команд содержит адрес A , содержимое ячейки с адресом A поступает в регистр команд и счетчик команд инкрементируется, принимая значение $A + 1$. Содержимое ячейки памяти по этому адресу поступает в регистр адреса. Счетчик команд инкрементируется еще раз, принимая

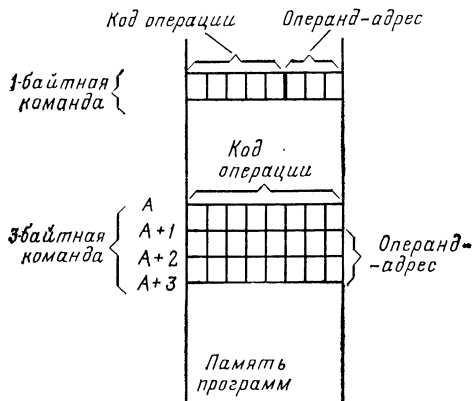


Рис. 2.5.

значение $(A + 2)$. Содержимое ячейки памяти по этому адресу также поступает в регистр адреса. После выполнения команды счетчик команд принимает значение $A + 3$, и выбирается следующая команда.

Для других команд и машин могут применяться другие правила, которые рассмотрим далее.

2.8. Арифметическо-логическое устройство

Арифметические и логические операции, которые должна выполнять микро-ЭВМ, осуществляются в арифметическо-логическом устройстве. Для большого числа арифметических операций требуются

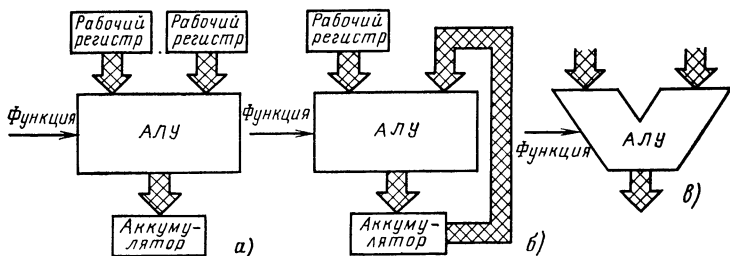


Рис. 2.6.

два операнда, например, сложить a и b . Операнды поступают в АЛУ из двух регистров общего назначения, и с ними выполняется операция; результат операции хранится в аккумуляторе. При таких операциях

ЭВМ должна выдавать адреса обоих операндов, поэтому говорят, что машина *двухадресная* (рис. 2.6, а).

Во многих микро-ЭВМ, однако, содержимое аккумулятора передается по цепи обратной связи на входы АЛУ. В этом случае один из операндов *всегда* находится в аккумуляторе, так что необходимо адресовать только второй операнд. Это — *одноадресная машина*.

П р и м е ч а н и е. Арифметическо-логическое устройство иногда изображают на схемах так, как это показано на рис. 2.6, б, из которого видно, что две части данных представлены одним результатом,

Выводы

1. Арифметические и логические операции выполняются в арифметическо-логическом устройстве. Результаты хранятся в аккумуляторе.

2. В двухадресной машине данные для операций можно получить из двух регистров общего назначения.

3. В одноадресной машине один из операндов находится в аккумуляторе.

2.9. Обмен данными в микро-ЭВМ

От ЦП к памяти. Операция записи в память выполняется для передачи данных из ЦП в выбранную ячейку памяти. Происходит следующее (рис. 2.7):

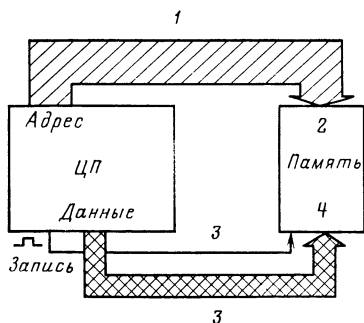


Рис. 2.7.

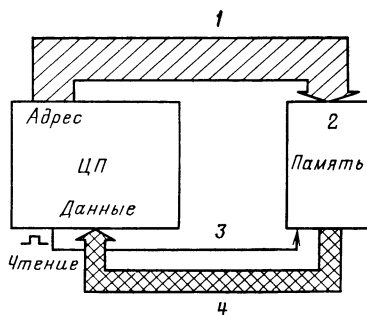


Рис. 2.8.

1) адрес ячейки памяти, в которую нужно записать данные, поступает из ЦП в память;

2) адрес дешифруется селектором адреса;

3) ЦП передает в память данные и одновременно управляющий сигнал *записи*;

4) данные заносятся в ячейку памяти по заданному адресу.

От памяти к ЦП. Операция считывания (чтения) из памяти выполняется для передачи данных, имеющихся в выбранной ячейке памяти, в ЦП. Происходит следующее (рис. 2.8):

1) адрес ячейки памяти, содержимое которой должно быть передано в ЦП, поступает из ЦП в память;

- 2) адрес дешифруется селектором адреса;
- 3) ЦП направляет в память сигнал считывания;
- 4) содержимое выбранной ячейки памяти поступает в ЦП.

Примечание. Команда, которая заставляет данные перемещаться в память или из памяти, называется командой обращения к памяти.

Из устройства ввода в ЦП. Команда ввода перемещает данные из устройства ввода в ЦП, где с данными проводятся дальнейшие операции. Передача данных из устройства ввода в ЦП осуществляется следующим образом (рис. 2.9):

1) ЦП направляет адрес выборки устройства ввода-вывода (УВВ) на модуль ввода-вывода через адресную шину и тем самым определяет, с какого порта ввода должны быть получены данные;

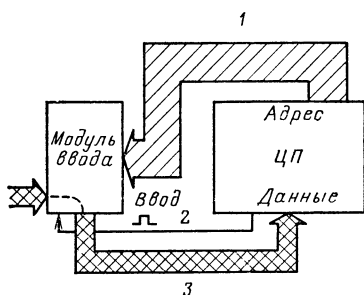


Рис. 2.9.

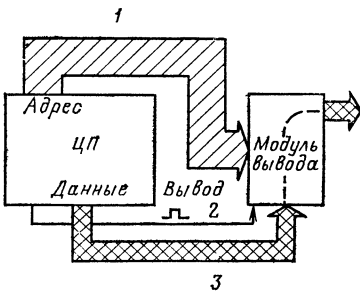


Рис. 2.10.

2) чтобы показать, что требуется операция ввода, ЦП посылает сигнал ввода модулю ввода-вывода через шину управления;

3) данные, представленные в выбранном порте ввода, переписываются в ЦП.

От ЦП на устройство вывода. Операция вывода выполняется для передачи данных от ЦП к выбранному порту вывода следующим образом (рис. 2.10):

1) адрес выборки УВВ, определяющий порт вывода, через который данные должны быть переданы на внешнее устройство, поступает с адресной шины на модуль ввода-вывода;

2) ЦП помещает данные для передачи на шину данных и, передавая сигнал вывода через шину управления, показывает, что данные готовы;

3) данные из ЦП через выбранный порт ввода передаются на внешнее устройство.

Выводы

1. Операция записи осуществляется для того, чтобы извлечь данные из ЦП и записать для хранения в выбранной ячейке памяти.

2. Операция считывания из памяти осуществляется для того, чтобы передать данные из памяти в ЦП.

3. Операция ввода осуществляется для того, чтобы передать данные в ЦП из выбранного порта ввода.

4. Операция вывода осуществляется для того, чтобы передать данные от ЦП на выбранный порт вывода.

2.10. Структура микро-ЭВМ

Структура микро-ЭВМ представлена на рис. 2.1. В нижнем правом углу имеется клавиатура для ввода данных и программ. Семисегментный светодиодный дисплей позволяет визуальным образом контролировать процесс ввода. На нем также отображаются результаты выполнения операций.

Если микро-ЭВМ используется для управления определенным процессом, то клавиатура и дисплей используются только для ввода программы и не нужны после запуска, т. е. после того, как программа начнет управлять системой.

Прямоугольник в левом нижнем углу рис. 2.1 — микропроцессор, в данном случае типа 8080. Генератор синхросигналов в этом типе ЭВМ расположен не на кристалле, а состоит из ИС 8224 и осциллятора, как показано в левой части рисунка.

Оперативное запоминающее устройство состоит из четырех ИС D5101. Эти четыре интегральные схемы содержат 512 слов памяти. На печатной плате есть место еще для четырех таких ИС.

Постоянное запоминающее устройство (ПЗУ) состоит из трех ИС D464. Это ПЗУ запрограммировано изготовителем. Рассмотрим его далее.

Модуль ввода-вывода образован одной ИС D8255. Эта интегральная микросхема содержит три порта, каждый из которых по желанию может использоваться как порт ввода или вывода.

С помощью выключателя *защита/разрешение* можно подсоединить несколько резервных батарей (аккумуляторов) к линии электропитания. Это делается для того, чтобы данные в памяти не пропали при отказе системы электропитания.

Глава третья

ОБЩИЕ СВЕДЕНИЯ О МИКРО-ЭВМ

3.1. Введение

Можно сказать, что четвертое поколение в истории электроники началось с появлением микропроцессора. Сначала были вакуумные лампы, затем транзисторы, еще позднее интегральные микросхемы; теперь же микропроцессоры открывают новый этап технологического развития. То, что это действительно новый этап, можно понять, если обратить внимание на то, что теперь имеем дело не с полупроводниковыми устройствами, а с системами. Применяя микропроцессор, можно создать систему, при этом важным является то, что это не просто процесс пайки элементов. Устанавливая компоненты на печатной плате и соединяя их, создаем *аппаратные средства* системы.

Чтобы система функционировала в соответствии с определенными требованиями, ее нужно *запрограммировать*,

т. е. обеспечить программой. Разработка такой программы заменяет проектирование схемы, необходимое в обычной системе.

В качестве примера на рис. 3.1 показаны две схемы, которые могут выполнять сложение. На рис. 3.1, а пока-

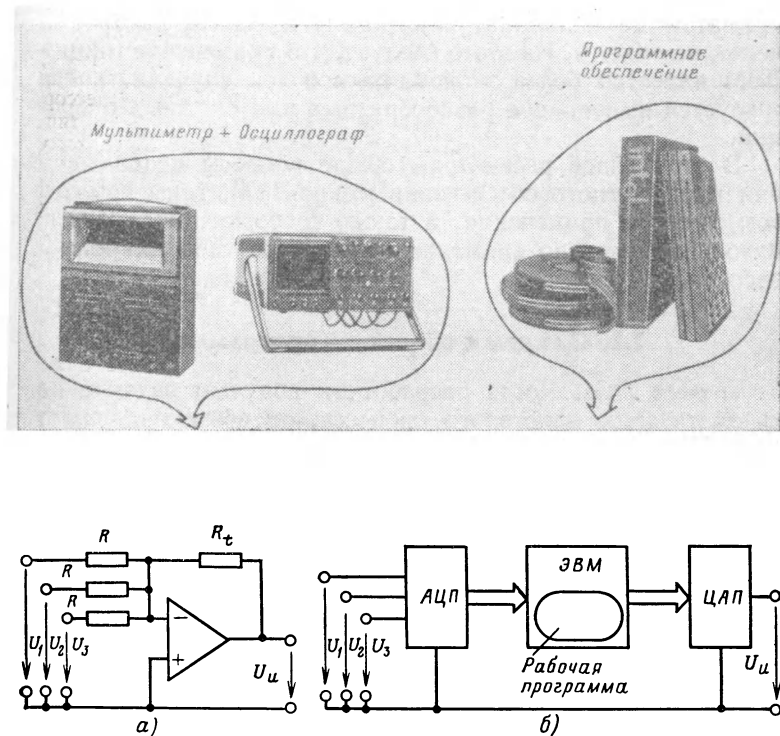


Рис. 3.1.

зана схема с операционным усилителем и резисторами; на рис. 3.1, б — система, в состав которой входит ЭВМ. Рабочая программа ЭВМ обеспечивает суммирование вводимых значений. Чтобы получить хорошо работающую программу, ее нужно подготовить и отладить.

Системы, в состав которых входят микро-ЭВМ, разрабатывают с использованием не обычных приборов, таких как мультиметр и осциллограф (рис. 3.1, а), а с использованием программ и подпрограмм (рис. 3.1, б). Эти подпро-

граммы, которые в комплексе называются *программным обеспечением*, предоставляются изготовителем микро-ЭВМ.

Преимущества вычислительных систем очевидны из приведенных примеров. Если нужно перемножить (или разделить) числа U_1 , U_2 , U_3 между собой, необходимо только ввести новую рабочую программу. Аппаратные средства при этом не меняются. Программы изменять быстрее и легче, чем схемы. Из этого следует, что применение микро-ЭВМ является более экономичным в тех случаях, когда требуется выполнение разнообразных или сложных операций.

В этой главе рассмотрим общие вопросы использования программного обеспечения микро-ЭВМ, типы памяти, возможности применения, а также соображения, которые надо принимать во внимание при выборе типа микропроцессора.

3.2. От идеи к объектной программе

Первая фаза. Когда разработчик получает задание на автоматизацию некоторого процесса, он обычно начинает с определения задачи. Затем он находит метод решения задачи и составляет алгоритм. В этом алгоритме последовательно указываются операции, которые должна выполнить ЭВМ. На основе алгоритма программист пишет программу или на языке низкого уровня (например, на языке ассемблера), или на языке высокого уровня (например, PL/M). Эта программа, написанная от руки, называется исходной программой.

Вторая фаза. Электронная вычислительная машина не может выполнить исходную программу, так как ее сначала необходимо перевести на язык машинных команд, т. е. в *объектную программу*. Для трансляции программы применяется *система разработки микропроцессора*. Система разработки — это специализированная микро-ЭВМ, используемая при разработке программного обеспечения.

Система разработки имеет память большой емкости для хранения подпрограмм и содержит необходимые периферийные устройства.

Для трансляции исходной программы в объектную ее необходимо сначала ввести в память системы разработки. Прежде всего печатают исходную программу на телетайпе. Каждая буква, число и цифра исходной программы становится символом, закодированным в ASCII, который хра-

няется в основной памяти системы разработки (подробное описание ASCII дано в гл. 4). Перемещение символов в системе разработки и хранение их в основной памяти осуществляется вспомогательной программой, которая хранится в памяти программ системы разработки и называется *ре-*

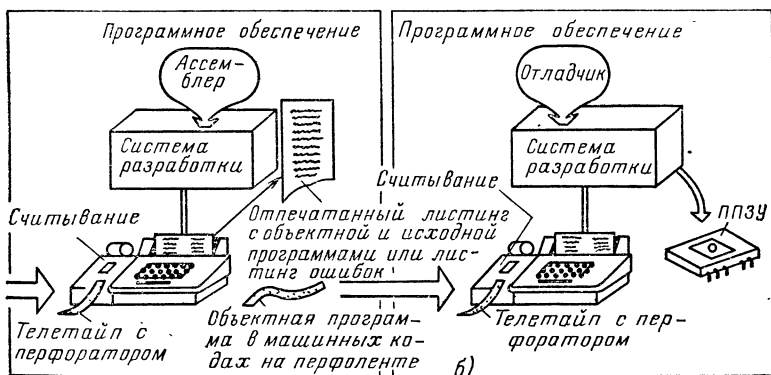
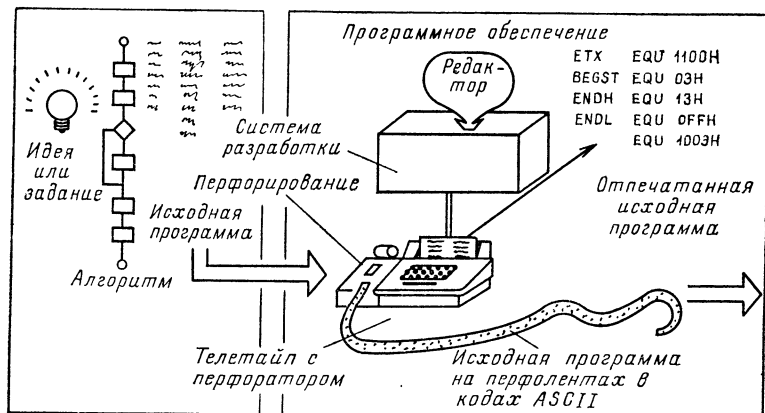


Рис. 3.2.

дактором. Опечатки могут исправляться редактором при необходимости.

Когда вся программа буква за буквой и цифра за цифрой будет занесена в память данных системы разработки, она выводится на перфоленту. Включают перфоратор телетайпа и дают команду системе разработки выдать исходную

программу в кодах ASCII на телетайп. Каждый символ теперь представлен в виде пробивок на перфоленте. Эта перфолента называется *исходной лентой*, так как содержит исходную программу.

Третья фаза. На этой стадии должна включиться в работу еще одна вспомогательная программа, так как микро-ЭВМ не может выполнить команды с исходной ленты, потому что на ней содержатся немашинные команды. При помощи программы транслятора (например, *ассемблера* для языка ассемблера или *компилятора* для языка высокого уровня) исходная программа преобразуется в системе разработки в машинные команды микро-ЭВМ. После того как транслятор введен в систему разработки, можно ввести исходную ленту, подготовленную во второй фазе разработки (информация, содержащаяся на исходной ленте, также хранится в памяти системы разработки), и начать трансляцию.

Выводы

1. Разработка программного обеспечения микро-ЭВМ осуществляется в системе разработки.

2. При разработке программного обеспечения используется несколько вспомогательных программ (системных программ), которые предоставляются изготовителем микро-ЭВМ.

3. Для ввода и исправления исходной программы используется вспомогательная программа — редактор.

4. Транслятор преобразует исходную программу в объектную.

5. Транслятор, работающий с программой, написанной на языке ассемблера, называется ассемблером.

Транслятор, работающий с программой, написанной на языке высокого уровня, называется компилятором.

3.3. От объектной программы к рабочей программе

Из рис. 3.2 видно, что в конце третьей фазы разработки система разработки выдает через телетайп следующий результат:

- а) листинг, содержащий введенную исходную программу, а также объектную программу в двоичных кодах;
- б) листинг, содержащий все обнаруженные ошибки;
- в) перфоленту, содержащую объектную программу, если исходная программа была без ошибок.

При первом просмотре результата третьей фазы разработки не нужно разочаровываться. В большинстве случаев список ошибок очень длинный и не получается перфолента с объектной программой. Нужно вернуться ко второй фазе разработки, где ошибки могут быть исправлены редактором. При этом содержимое ячеек памяти, в которых хранятся символы ASCII, будут записаны заново. После этого нужно сделать новую исходную перфоленту и повторно ввести ее в систему разработки. Этот процесс повторяется до тех пор, пока не исчезнут ошибки. Наконец появляется следующий текст: «ассемблирование выполнено — ошибок нет». Это означает только то, что транслятор ошибок не обнаружил, но совсем не означает, что полученная программа работоспособна.

Четвертая фаза. Теперь, когда получена перфолента, содержащая объектную программу, ее нужно ввести в память системы разработки, чтобы посмотреть, будет ли она работать. В большинстве случаев программа с первой попытки не работает, и нужно найти ошибки. Они обычно носят фундаментальный характер, например: ошибка в алгоритме, использование неправильной команды и т. д.

Если не работает обычная схема, то необходимо взять паяльник и осциллограф. В системе разработки используют инструмент программирования — *программу-отладчик* или *дебаггер*¹. Отладчик (который помогает «вылавливать блох» из программы) позволяет обнаружить слабые места в программе, используя телетайп вместо осциллографа. Отладчик вводится в память системы разработки, в которой уже хранится объектная программа, и дается команда типа: «Исполнить программу от точки *n* до точки *m*. Затем остановиться и сообщить, что хранится в памяти». Телетайп быстро выдает ответ.

Таким образом можно постепенно просмотреть всю программу и проверить содержимое всех регистров, ячеек памяти и т. д. Каждый сегмент программы может быть отдельно изучен и исправлен. Наконец, получаем объектную программу без ошибок или рабочую программу.

Пятая фаза. На последней фазе разработки программа вводится в ППЗУ, входящее в состав микро-ЭВМ, для которой разработана программа.

¹ debugger (англ.) — вылавливатель блох. (Прим. пер.)

3.4. Типы памяти

Основная память системы ЭВМ состоит из памяти программ и памяти данных (рис. 3.3). Команды хранятся в памяти программ, а данные для обработки — в памяти данных. Постоянное запоминающее устройство обычно используется как память программ, а ОЗУ — для хранения данных. Оба типа памяти выполнены на полупроводниковых элементах.

Информация (чаще всего программа) постоянно хранится в ПЗУ. Ее можно только считывать и нельзя менять или обновлять. Имеются три типа ПЗУ:

1) ПЗУ, запрограммированное изготовителем микро-ЭВМ.

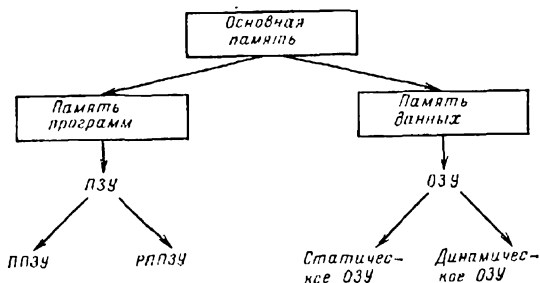


Рис. 3.3.

Во время производства ПЗУ изготовитель вводит в память информацию в соответствии с требованиями пользователя; последующее обновление информации невозможно. Этот тип ПЗУ используется, когда необходимо большое число одинаковых ПЗУ;

2) программируемые ПЗУ (ППЗУ).

Пользователь может запрограммировать ППЗУ с помощью программирующего устройства, которое выжигает адресуемые диоды в матрице ППЗУ; после этого дальнейшие изменения содержимого памяти невозможны. Этот тип памяти используется, когда требуется небольшое число различных ПЗУ;

3) РППЗУ (репрограммируемые ППЗУ) или стираемые ППЗУ.

В этих устройствах информация может стираться несколько раз с помощью ультрафиолетового облучения. Перепрограммирование осуществляется с помощью програм-

мирующего устройства ППЗУ. Преимущество РППЗУ заключается в том, что облегчается исправление ошибок и можно производить изменение содержимого памяти, не выбрасывая ПЗУ или ППЗУ.

Оперативное запоминающее устройство — это память, из которой процессор может считывать или в которую может записывать информацию. Поэтому ее используют для хранения промежуточных результатов вычислений и переменных, и для микро-ЭВМ она является своего рода блоком для записей.

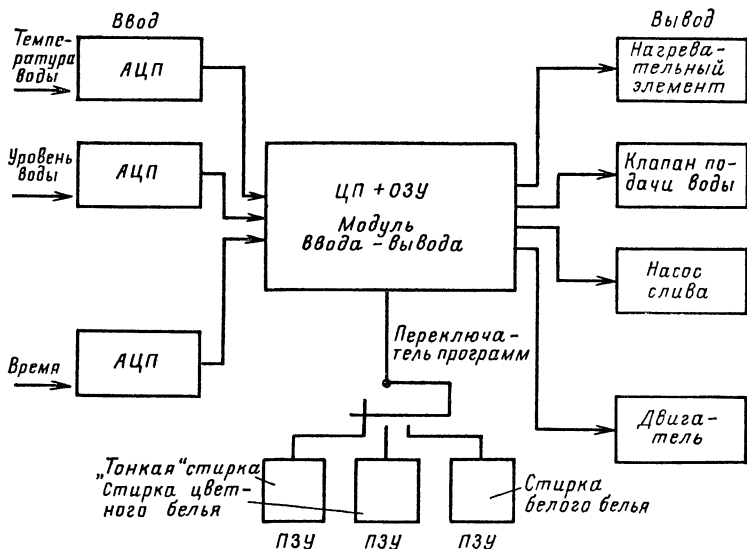


Рис. 3.4.

П р и м е ч а н и е. В зависимости от принципа хранения информации различают статические и динамические схемы ОЗУ. Обычно в микро-ЭВМ применяются статические ОЗУ.

Выводы

1. Объектная программа, появившаяся в третьей фазе разработки, заносится в память системы разработки.

2. Для обнаружения и исправления ошибок в объектной программе используется программа-отладчик.

3. Окончательным результатом разработки является рабочая программа, которая помещается в ППЗУ.

4. Постоянное ЗУ в основном используется как память программ, ОЗУ — как память данных.

5. Программируемое ПЗУ — это тип памяти, которую пользователь программирует 1 раз. Репрограммируемое ППЗУ является запоминающим устройством, в котором

храняемая информация может быть стерта с помощью ультрафиолетового облучения.

6. Различают два типа ОЗУ: статические и динамические.

3.5. Пример применения микро-ЭВМ

Для демонстрации работы микро-ЭВМ рассмотрим простой пример применения микро-ЭВМ в системе управления стиральной машиной (рис. 3.4).

Вводимые данные, такие как температура и уровень воды в баке, должны быть преобразованы с помощью АЦП в сигналы, воспринимаемые ЦП. Сигналы, выдаваемые ЦП, управляют нагревательными элементами, клапаном подачи воды, насосом слива воды, мотором для стирки и центрифугой. Управление осуществляется с помощью реле, трехпозиционных управляемых переключателей и т. д.

В ПЗУ хранятся программы различных режимов стирки. Изготовителю легко увеличить число этих программ или изменить программу путем замены одной или нескольких БИС ПЗУ. Для демонстрации процесса управления на

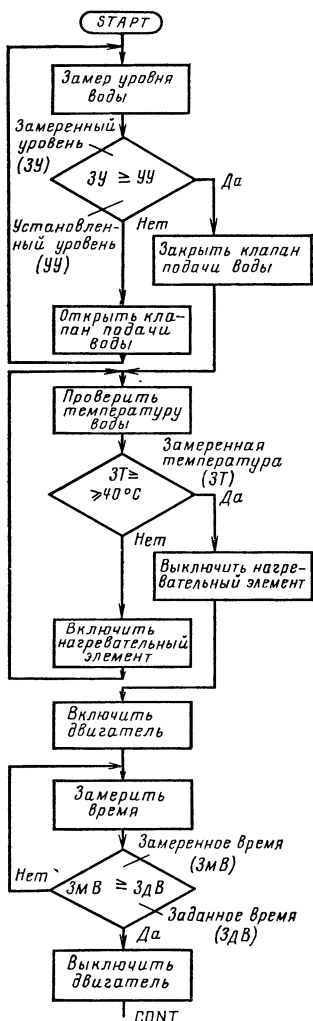


Рис. 3.5.

Сфера потребления	Автоматические бензоколонки, кассовые аппараты, автоматические разменные аппараты, игрушки, бильярдные машины, кабельное телевидение, автомобили, кухонные СВЧ духовые шкафы, швейные машины, радио, телевидение
Медицина/биология	Диагностические устройства, лабораторные анализы, уход за больными, системы физиологических исследований, организация приема больных
Связь	Передающее оборудование, управление каналами, подстройка
Физические приборы	Управление реакторами, масс-спектрометры, системы сигнализации загрязнения воздуха
Измерительные приборы	Осциллографы, анализаторы спектра, системы контроля компонентов, системы сортировки, весы, радиолокационные устройства, контрольно-измерительная аппаратура, дозирочные системы
Бортовые ЭВМ	Для самолетов, судов, грузовых автомобилей, военных систем, ракетных систем (воздушных и космических)
Управление в промышленности	Управление процессом, регулирование уличного движения, управление подъемниками, автономные системы управления (подсистемы), системы роботов, управление машинами, управление производством, управление типографскими машинами, центральный узел локальной системы управления, система контроля ИС и печатных плат
Периферийные устройства ЭВМ	Дисплей, устройства считывания с перфокарт, гибкие диски, управление НМЛ и кассетными НМЛ, печатающие устройства
Обработка данных	Интеллектуальные терминалы, устройства подготовки счетов, запоминающие устройства, мультимикропроцессоры (системы, состоящие из множества МП), карманные калькуляторы, банковские устройства (проверка чеков и т. д.), конторское оборудование (организация картотеки), автоматические пишущие машинки, фотокопировальные машины

рис. 3.5 приведена часть блок-схемы программы «тонкой» стирки. Эта программа постоянно хранится в ПЗУ «тонкой» стирки. Выделенные на рис. 3.5 параметры постоянно хранятся в ПЗУ: температура воды, ее уровень, время, необходимое для стирки и сушки. Каждый раз, когда измеряется уровень воды, ее температура или время работы электропривода (для стирки или сушки), микро-ЭВМ должна принимать решение путем сравнения замеренных значений параметров со значениями, хранящимися в ПЗУ. Например, при «тонкой» стирке температура воды не должна превышать 40 °С. Если при замере температура воды оказывается равной или большей 40 °С, то нагревательный элемент выключается и включается клапан подачи воды.

3.6. Области применения микро-ЭВМ

В табл. 3.1 перечислены области применения микро-ЭВМ.

3.7. Выбор типа микро-ЭВМ

Решив использовать в разработке микро-ЭВМ, необходимо выбрать ее тип, который соответствует требованиям разрабатываемой системы. Это сделать не просто, так как существует много разнообразных микро-ЭВМ, а также факторов, которые необходимо учитывать при принятии решения.

Т а б л и ц а 3.2

-
- | | |
|-----|--------------------------------------|
| 1. | Наличие изготовителя |
| 2. | Уверенность в изготовителе |
| 3. | Поддержка изготовителя |
| 4. | Набор команд |
| 5. | Быстродействие |
| 6. | Архитектура |
| 7. | Методы адресации |
| 8. | Число источников электропитания |
| 9. | Потребление электроэнергии |
| 10. | Обслуживаемость |
| 11. | Разнообразие элементов семейства БИС |
| 12. | Документация |
-

Эти факторы принято подразделять на технические и конъюнктурные. Опрос мнения пользователей микро-ЭВМ, проведенный одной из ведущих фирм — изготовителей микро-ЭВМ в США, показал, что основную роль при выборе типа микро-ЭВМ играют не технические, а конъюнктурные факторы (табл. 3.2). Рассмотрим подробнее некоторые из них.

Наличие не одного, а хотя бы двух независимых изготовителей микро-ЭВМ является очень важным фактором.

Поддержка изготовителя (поставка готовых подпрограмм и при необходимости оказание помощи в разработке рабочих программ) также влияет на решение о выборе типа микро-ЭВМ.

Очень важен набор команд микро-ЭВМ. Нужно представить себе, можно ли необходимую операцию выполнить с использованием одной или нескольких команд.

Скорость иногда очень важна в управлении процессом, и высокие требования к быстродействию системы могут не позволить использование микро-ЭВМ.

При выборе типа микро-ЭВМ следует также учитывать ее архитектуру. Важное значение при этом имеет структура шин, организация памяти и возможность расширения системы.

В гл. 13 рассмотрены системы адресации операндов. Часто можно значительно сократить емкость памяти путем правильного выбора техники адресации. Поэтому если МП может использовать различные способы адресации, то это может дать определенные преимущества.

Выбирая микропроцессор, нужно внимательно изучить и другие элементы, входящие в состав семейства. В том случае, например, если определенный тип МП не имеет сопрягаемых модулей ввода-вывода, необходимо применять модули другого типа, что обычно вызывает дополнительные проблемы (разница в напряжении, скорости и т. д.).

В табл. 3.2 показано, что такие аспекты, как уверенность в изготовителе и наличие поддержки изготовителя микро-ЭВМ, оказывают большее влияние на выбор типа микро-ЭВМ, чем их технические характеристики.

Глава четвертая

ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В ЭВМ

4.1. Введение

Цифровые электронные устройства, к которым относятся и цифровые вычислительные машины, строятся на схемах, способных находиться лишь в двух состояниях. Эти схемы можно представить в виде набора выключателей, причем каждый выключатель может быть открыт или закрыт для протекания электрического тока. Состояниям выключателя, в которых он проводит или не проводит ток, приписывают соответственно символы 1 или 0. Совокупность состояний, характерная для определенного набора ключей в некоторый момент времени, может рассматриваться как последовательность 0 и 1. Если существует соглашение о способе кодирования информации, то такая последовательность 0 и 1 может представлять число, букву или какой-нибудь другой символ алфавита вычислительной машины. Представление чисел с помощью всего лишь двух символов: 0 и 1 — двоичная система счисления, или система счисления с основанием 2 ($bi = 2$). Каждый разряд двоичной записи числа называется *битом* (сокращение английских слов binary digit, означающих «двоичная единица»).

В данной главе рассматриваются операции над двоичными числами. В повседневной жизни для человека более привычна десятичная система счисления (система счисления с основанием 10), и поэтому чтобы научиться свободному обращению с двоичными числами, требуется некоторое время. В действительности же правила вычислений в двоичной системе счисления значительно проще, чем в десятичной: легче оперировать двумя цифрами (0 и 1), чем десятью (0—9), которые входят в алфавит десятичной системы.

Отрицательные двоичные числа в вычислительной машине представляют в дополнительных кодах, так как набор символов, с помощью которых записываются числа, ограничен 0 и 1.

В настоящей главе рассматривается шестнадцатиричная система счисления, которую можно рассматривать как метод сокращенного представления последовательностей из 0 и 1.

Для представления алфавитно-цифровой информации в двоичной форме существуют различные коды. Рассмотрим два кода, применяемые в микро-ЭВМ: ASCII и EBCDIC.

4.2. Десятичная система счисления

Для лучшего понимания двоичной системы счисления сначала повнимательнее рассмотрим хорошо знакомую десятичную систему.

Алфавит десятичной системы счисления состоит из десяти символов: 0, 1, ..., 9. Эти символы используются для записи чисел или значений физических величин. Числа больше девяти представляют в виде последовательности цифр. Эта последовательность строится не случайным образом, а в соответствии с определенными правилами.

Последовательность цифр, изображающая некоторое число, строится таким образом, что позиция каждой цифры в последовательности имеет определенный вес. Например, в числе 247 самая значащая цифра — 2. Можно сказать, что в этом числе цифра 2 имеет наибольший вес.

Весом называется коэффициент, на который следует умножить цифру для нахождения ее истинного значения в последовательности, изображающей число.

В десятичной записи числа самая правая цифра имеет вес 1, соседняя слева цифра имеет вес 10, следующая — 100 и т. д.

Можно записать число как сумму степеней десяти. Например, $247 = 2 \cdot 100 + 4 \cdot 10 + 7 \cdot 1 = 200 + 40 + 7 = 2 \cdot 10^2 + 4 \cdot 10^1 + 7 \cdot 10^0$.

В зависимости от веса позиции, в которой стоит цифра, различают разряды единиц, десятков, сотен и т. д.

З а м е ч а н и е. Единицу можно записать в виде 10^0 , так как по определению нулевая степень любого числа есть единица.

Выводы

1. Число состоит из цифр, каждая из которых имеет определенный вес.

2. *Весом* цифры называют коэффициент, на который следует умножить цифру для нахождения ее истинного значения.

3. Алфавит двоичной системы счисления, или системы счисления с основанием 2 ($bi = 2$), состоит из цифр 0 и 1.

4. Слово *bit* — сокращение английских слов binary digit.

5. По определению нулевая степень любого числа есть единица: $10^0 = 1$.

4.3. Двоичная система счисления

Производя вычисления над десятичными числами, поступают следующим образом (рис. 4.1.).

При сложении числа с единицей цифру в первом (самом правом) разряде заменяют следующей, старшей по значению цифрой. Например, при сложении единицы и 38 результатом будет 39. Первый разряд теперь целиком заполнен, так как девять — это наибольшее однозначное число.

При повторном сложении с единицей первый разряд обращается в 0, а в соседний слева второй разряд добавляется 1 (единица переноса в старший разряд). Во втором разряде появляется цифра 4. Будем

добавлять по единице к первому разряду до тех пор, пока он не примет максимальное значение (9). Следующее сложение с единицей приведет к появлению единицы переноса во второй разряд.

Таким образом, цифра во втором разряде показывает, сколько раз первый разряд заполнялся целиком до максимального значения стоящей в нем цифры.

В двоичной системе счисления располагают лишь двумя цифрами: 0 и 1, и поэтому при сложении чисел в двоичной системе разряды заполняются очень быстро (табл. 4.1). Первый разряд заполняется сразу же после сложения с единицей. После повторного добавления единицы, результату которого соответствует десятичное число два, первый разряд обращается в нуль и производится перенос из первого разряда во второй. При третьем добавлении единицы первый разряд снова обращается из нуля в единицу и оба разряда оказываются заполнены. После четвертого добавления единицы, результату которого соответствует число четыре, первый разряд обращается в нуль. Перенос единицы во второй разряд, который был уже заполнен, обращает этот разряд в нуль и приводит к необходимости переноса единицы в третий разряд. Повторяя сложение с единицей описанным образом, можно получить все остальные двоичные числа.

Очевидно, что двоичная и десятичная системы счисления в высшей степени схожи. Так же, как и в десятичной системе счисления, цифра в каждом разряде двоичного числа показывает, сколько раз заполнялся до максимума разряд, стоящий справа.

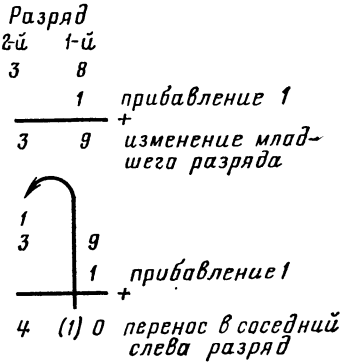


Рис. 4.1.

Таблица 4.1

Число		Число	
десятичное	двоичное	десятичное	двоичное
0	0	11	1011
1	1	12	1100
2	10	13	1101
3	11	14	1110
4	100	15	1111
5	101	16	10000
6	110	17	10001
7	111	18	10010
8	1000	19	10011
9	1001	20	10100
10	1010		

4.3.1. Преобразование чисел из двоичной системы в десятичную

Так как вес каждого разряда двоичного числа вдвое больше веса разряда стоящего справа, двоичное число можно представить в виде суммы степеней числа 2. Например, $101101 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 32 + 0 + 8 + 4 + 0 + 1 = 45$.

Отсюда видно, что запись числа в виде суммы степеней числа 2 позволяет получить его десятичный эквивалент.

П р и м е ч а н и е 1. Последовательность цифр 100 имеет два различных значения:

а) в десятичной записи эта последовательность цифр представляет собой число сто;

б) в двоичной записи эта последовательность цифр представляет собой значение, эквивалентное десятичному числу 4.

В тех случаях, когда не будет достаточно очевидно, какая именно система счисления использована для представления числа, будем записывать основание системы в скобках справа от числа под строкой как индекс, например $100_{(2)}$. Двоичная запись числа $1001_{(2)}$ читается как «один, ноль, ноль, один».

Десятичные числа преобразуют в двоичные следующим образом. Для перевода $31_{(10)}$ (31 в десятичной записи) в двоичную систему счисления следует записать 31 как сумму степеней числа 2. Для этого сначала определим, чему равна наибольшая степень числа 2, содержащаяся в исходном числе. Для 31 это $16 = 2^4$. Найдем остаток от вычитания 16 из 31 (15) и, в свою очередь, вычтем из него следующую наибольшую степень числа 2, т. е. $8 = 2^3$ и т. д. Окончательно получим:

$$31_{(10)} = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0.$$

Располагая коэффициенты при степенях числа два в порядке их определения, получаем искомое двоичное представление числа.

П р и м е ч а н и е 2. Самый левый разряд двоичной записи числа часто называют старшим значащим разрядом (разрядом с наибольшим весом). Самый правый разряд при этом называют младшим значащим разрядом (разрядом с наименьшим весом).

Выводы

1. Для записи двоичных чисел используют две цифры: 0 и 1.
2. Двоичное число можно преобразовать в десятичное, представив его в виде суммы степеней числа 2.
3. Десятичное число преобразуется в двоичное путем последовательного вычитания степеней числа 2, начиная с наибольшей степени, содержащейся в числе. Найденные в результате коэффициенты располагают в последовательность таким образом, чтобы коэффициент при старшей степени числа 2 стоял в разряде с наибольшим весом.
4. Основные системы счисления указывают в виде индекса, взятого в скобки.

4.4. Сложение двоичных чисел

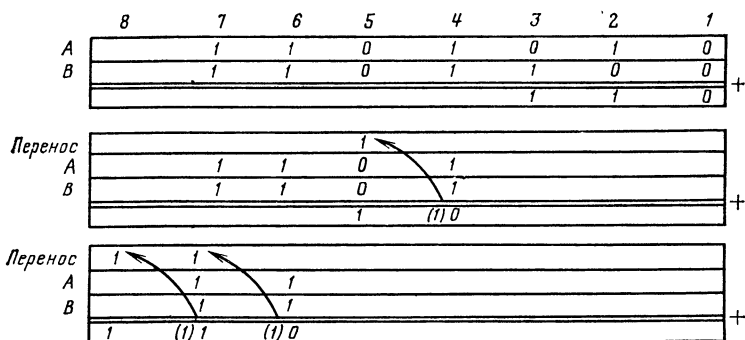
Правила сложения двоичных и десятичных чисел аналогичны, но в результате более быстрого заполнения разрядов в двоичной системе (табл. 4.2) быстрее происходит и перенос в старший разряд при сложении двоичных чисел.

Прибавление единицы к старшему разряду в результате переполнения соседнего, младшего разряда называют переносом.

Как видно из рис. 4.2, сложение двоичных чисел A и B выполняется по тем же правилам, что и сложение десятичных чисел. В первом разряде слагаемыми являются 0 и 0, результат получается 0. Во втором разряде к 1 прибавляется 0, результат получается 1. В третьем разряде к 0 прибавляется 1, результат получается 1. В четвертом разряде результатом сложения 1 с 1 является 10. Единицу переноса записываем над пятым разрядом, в котором суммирование 1, 0 и 0 дает в результате 1.

Таблица 4.2

Число	
десятичное	двоичное
$0+0=0$	$0+0=0$
$0+1=1$	$0+1=1$
$1+1=2$	$1+1=10$
$1+1+1=3$	$1+1+1=11$



слева разряду, видим, что в нем нужно вычесть 8 из 0. Непосредственно это сделать невозможно, так как $8 > 0$ или, другими словами, нельзя уменьшить 0 на 8. Чтобы продолжить вычисления, необходимо обратиться к разрядам, стоящим слева, для нахождения числа, не равного 0. В данном случае таким числом будет 9. Занимаем 1 из 9, в результате чего в соответствующем разряде вместо 9 появляется 8 (рис. 4.3, б). Теперь к 0 во втором разряде можно добавить 1, что дает в нем число 10.

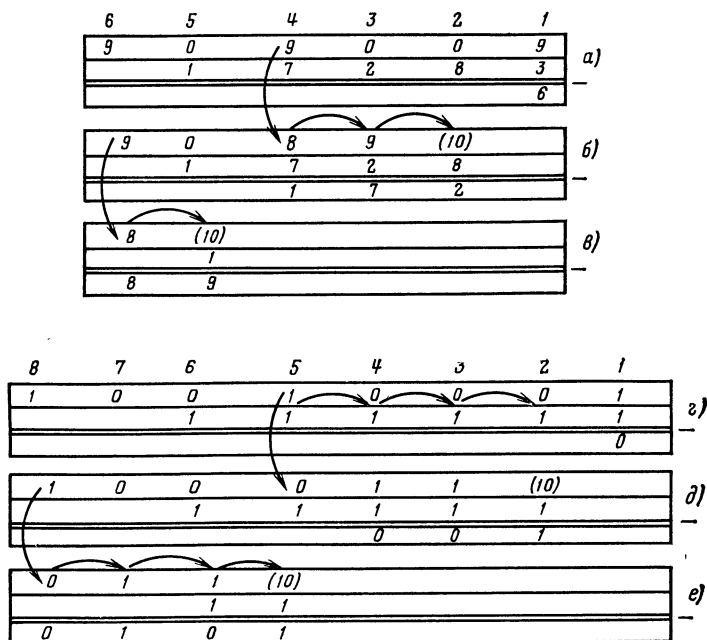


Рис. 4.3.

Нули, стоящие между разрядами, которые связаны с займом единицы, заменяют на десятки (в рассматриваемом случае происходит лишь одна такая замена).

Теперь во втором разряде из 10 можно отнять 8, получив 2. В третьем разряде нужно вычесть 2 из 9, получим 7. В четвертом разряде вычитаем 7 из 8, получаем 1. В пятом разряде, в котором нужно вычесть 1 из 0, снова сталкиваемся с трудностями. Нужно опять занять единицу, двигаясь влево до тех пор, пока не дойдем до первого ненулевого разряда. В нем 9 заменяем на 8, а вместо 0 в пятом разряде получаем 10. Нулей между двумя разрядами, которые следовало бы заменить на девятки, в данном случае нет. Теперь вычисления можно продолжить (рис. 4.3, е).

Вполне возможно, что никогда ранее никто не обращал внимания на то, что при вычитании действительно нужно выполнять все эти действия. Теперь же, освежив в памяти необходимые правила, разо-

браться с вычитанием двоичных чисел будет проще. Сделаем это на следующем примере (рис. 4.3, г).

Результат в самом правом разряде находим без труда: $1 - 1 = 0$. Во втором разряде вычесть 1 из 0 так просто не удастся, и поэтому просматриваем разряды справа налево до тех пор, пока не найдем 1. Добавляем эту 1 к 0 во втором разряде, что дает в нем 10 (один, нуль). Числу $10_{(2)}$, как известно, соответствует десятичное число 2. Нули которые стоят между двумя разрядами, обращаются в единицы (рис. 4.3, д). Продолжим вычитание в третьем и четвертом разрядах с учетом того, что $10_{(2)} - 1_{(2)}$ равно $1_{(2)}$. В пятом разряде опять нужно вычесть 1 из 0, для чего производим заем, двигаясь влево до тех пор, пока не дойдем до 1. Эту 1 добавляем к 0 в пятом разряде и получаем в нем 10. Нули, стоящие между разрядами, заменяем на единицы, после чего вычитание можно продолжить (рис. 4.3, е).

<i>Десятичные числа</i>	<i>Двоичные числа</i>
<p><i>Вычитаемое</i> 19 31</p> $\begin{array}{r} 19 \\ 31 \\ \hline - 12 \\ \hline \end{array}$	$\begin{array}{r} 10011 \\ 11111 \\ \hline - 01100 \\ \hline \end{array}$

Рис. 4.4.

В рассмотренном примере уменьшаемое было больше вычитаемого и поэтому в качестве разности получилось положительное число. Однако в том случае, когда вычитаемое больше уменьшаемого, результат окажется меньше 0. Далее показано, как происходит вычитание в таком случае (рис. 4.4).

Очевидно, что в этом случае нужно сначала поменять местами уменьшаемое и вычитаемое, а затем выполнить вычитание и поставить знак минус перед результатом.

Выводы

1. *Переносом* называют прибавление единицы к следующему разряду.
2. Если нужно занимать при вычитании, то разряд, из которого занимают, обращается из 1 в 0; разряд, в который помещают, получает значение 10, а все промежуточные нули заменяют на единицы.
3. Если вычитаемое больше уменьшаемого, то вычитают уменьшаемое из вычитаемого, а перед разностью ставят знак минус.

4.6. Дополнительный код двоичного числа

То обстоятельство, что вычислительная машина оперирует лишь нулями и единицами, а знаки плюс и минус при этом отсутствуют, накладывает определенные ограничения на выполнение арифметических действий. Поэтому для того, чтобы можно было работать не только с положительными, но и с отрицательными числами, необходимо договориться о методе представления чисел, меньших нуля. Познакомим читателя с наиболее распространенным методом, который рассмотрим на примере 8-разрядного числа. Большинство микро-ЭВМ работают со словами именно такой длины.

В самом общем виде 8-разрядное двоичное число можно представить, как показано на рис. 4.5, причем b_0, \dots, b_7 принимают значения 0 или 1.

Десятичный эквивалент такого числа всегда положителен. Возможность оперировать как положительными, так и отрицательными числами появляется, если число представить следующим образом:

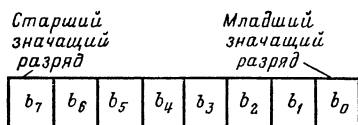


Рис. 4.5.

не 2^7 , а $-2^7 = -128$. Если в этом разряде стоит 1, то он будет представлять десятичное значение -128 . Число же в целом окажется отрицательным, так как наибольшее число, содержащееся в разрядах b_0, \dots, b_6 , не может превысить десятичное значение 127.

Таким образом, например, дополнительный код двоичного числа 100110001 можно представить как

$$1 \cdot (-128) + 0 \cdot 64 + 1 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = -79.$$

Если в старшем значащем разряде стоит 0, подобная запись представляет собой положительное число, так как у всех остальных разрядов b_6, \dots, b_0 веса положительны. Например,

$$01001100 = 0 \cdot (-128) + 0 \cdot 64 + 1 \cdot 32 + 0 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 0 \cdot 1 = 36.$$

Рассмотренный метод представления положительных двоичных чисел оказывается весьма простым: число записывается как сумма степеней числа 2, причем выбор длины разрядной сетки должен обеспечивать равенство нулю старшего значащего разряда. Представление отрицательного числа в виде дополнительного кода также строится достаточно просто.

Построим, например, дополнительный код числа 77. Для этого сначала запишем 77 в двоичной форме, после чего в двоичной записи заменим единицы на нули, а нули на единицы, т. е. образуем дополнение числа до единицы. Прибавление к получившемуся числу единицы даст искомый код:

$$\begin{array}{r} 77 = 01001101 \\ \quad 10110010 \\ \hline -77 = 10110011 \end{array} \quad \begin{array}{l} \text{дополнение до единицы} \\ \text{дополнительный код} \end{array}$$

Восемь двоичных разрядов позволяют представить числа в диапазоне от $10000000 = -128$ до $01111111 = +127$ (табл. 4.3). Если такой диапазон окажется слишком малым для представления исходных данных или результатов вычислений, то следует использовать 16-разрядные числа, которые образуются путем объединения двух 8-разрядных слов. Результат такого объединения (16-разрядное слово) называют словом двойной длины (рис. 4.6).

При использовании слов такой длины старшему значащему разряду следует присваивать вес $-(2)^{15}$. Наибольшим по модулю отрицательным числом, которое можно представить, будет

$$1\ 000\ 0000\ 0000\ 0000 = -(2)^{15} + 1 \cdot 2^{+14} + \dots + 1 \cdot 2^0 = -32\ 768.$$

Таблица 4.3

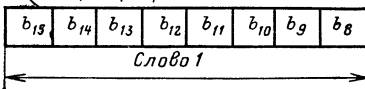
Число		Число	
двоичное	десятичное	двоичное	десятичное
10000000	—128	00000001	1
10000001	—127	00000010	2
10000010	—126	00000011	3
10000011	—125	⋮	⋮
⋮	⋮	01111101	+125
11111110	—2	01111110	+126
11111111	—1	01111111	+127
00000000	0		

Наибольшее положительное число

$$0111\ 1111\ 1111\ 1111 = 0 \cdot [-(2)^{15}] + 1 \cdot 2^{14} + \dots + 1 \cdot 2^0 = 32\ 767.$$

Использование дополнительных кодов двоичных чисел значительно облегчает выполнение операции двоичного вычитания.

Старший
значащий разряд



Младший
значащий разряд

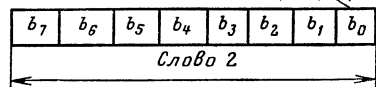


Рис. 4.6.

Очевидно, что вычитание и сложение являются, по существу, одной и той же операцией. Например,

$$17 - 22 = 17 + (-22).$$

Это означает, что вычитание $00010110_{(2)}$ из $00010001_{(2)}$ эквивалентно сложению $00010001_{(2)}$ с дополнительным кодом числа $00010110_{(2)}$. Вычисления проводятся в следующей последовательности.

Дополнительный код числа $00010110_{(2)}$ находим, сначала заменяя числа всех разрядов на противоположные и добавляя затем единицу (рис. 4.7).

$$\begin{array}{rcl}
 & \text{Обращение значений} & \\
 & \text{разряда} & \\
 00010110 & 11101001 & \text{Обратный код} \\
 & \underline{1} & + \\
 & 11101010 & \text{Дополнительный} \\
 & & \text{код}
 \end{array}$$

Рис. 4.7.

После этого $00010001_{(2)} - 00010110_{(2)}$ можно записать как

$$\begin{array}{r}
 00010001 \\
 11101010 \\
 \hline
 11111011
 \end{array}$$

Представление двоичных чисел с помощью дополнительных кодов применяется в очень многих микро-ЭВМ. Например, если используется команда SUB (вычитание), то в микро-ЭВМ числа, вычитаемые одно из другого, будут представлены в виде дополнительных кодов. Если же в микро-ЭВМ не предусмотрено автоматическое представление чисел в виде дополнительных кодов, то программист может сначала использовать команду, преобразующую вычитаемое в дополнительный код, а после этого выполнить сложение с уменьшаемым с помощью команды ADD (сложение).

4.7. Умножение двоичных чисел

Таблица двоичного умножения очень проста, как это видно из помещенных в табл. 4.4 всех возможных комбинаций перемножаемых.

Т а б л и ц а 4.4. Правила
двоичного умножения

$$\begin{aligned} 0 \cdot 0 &= 0 \\ 0 \cdot 1 &= 0 \\ 1 \cdot 0 &= 0 \\ 1 \cdot 1 &= 1 \end{aligned}$$

Рассмотрим пример:

$$\begin{array}{r} 1001 \\ 11 \quad + \\ \hline 1001 \\ 1001 \quad + \\ \hline 11011 \end{array}$$

Он показывает, что перемножение чисел в двоичной системе счисления выполняется так же, как в десятичной. При перемножении двоичных чисел, содержащих более двух единиц, сложение производится по шагам:

$$\begin{array}{r} 1101 \\ 1011 \quad + \\ \hline 1101 \quad + \\ 1101 \quad + \\ \hline 100111 \\ 1101 \quad + \\ \hline 1000111 \end{array}$$

Результаты умножения на ноль всегда есть ноль, поэтому такие результаты учитываются лишь сдвигом следующего слагаемого на одну позицию влево. При умножении на единицу, как и в десятичной системе счисления, получается то же самое число.

4.8. Терминология

Один из недостатков двоичной системы счисления заключается в том, что для записи двоичных чисел требуется больше разрядов, чем для эквивалентных десятичных чисел. Для удобства обращения

с такими «длинными» числами употребляется набор специальных терминов. Среди них прежде всего укажем на слово «бит», которое, как упоминалось ранее, происходит от сокращения слов binary digit и означает «двоичная единица». Тогда двоичное число 10111 оказывается состоящим из 5 бит, причем двоичное число в целом называют словом.

Важной характеристикой вычислительной машины является используемая в ней длина слова. Длинной слова называют число содержащихся в нем бит.

Последовательность из 4 бит называют *тетрадой* или *слогом*. Последовательность из 8 бит называют *октетом* или *байтом*.

4.9. Шестнадцатиричная система счисления

Шестнадцатиричная система счисления, или система счисления с основанием 16, также применяется в цифровой вычислительной технике, но не при выполнении вычислений, а для сокращенной записи двоичных чисел. Другими словами, шестнадцатиричная система используется как средство кодирования двоичных чисел.

Алфавит шестнадцатиричной системы счисления состоит из 16 символов: 0, ..., 9, A, B, C, D, E, F. В табл. 4.5 приведены изображения некоторых двоичных, десятичных и шестнадцатиричных чисел. Заметим, что для представления числа в шестнадцатиричной форме требуется меньшее число цифр.

Таблица 4.5

Число			Число		
десятичное	двоичное	шестнадцатиричное	десятичное	двоичное	шестнадцатиричное
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Правила перевода чисел из двоичной системы счисления в шестнадцатиричную и обратно весьма просты. Для записи двоичного числа в шестнадцатиричном виде сначала следует разбить исходное число на группы из 4 бит, т. е. на тетрады, каждой из которых соответствует шестнадцатиричный символ. Например, разбиение слова $101100101001_{(2)}$ на тетрады дает 1011 0010 1001; результатом замены тетрад соответствующими шестнадцатиричными символами будет искомое представление $B29_{(16)}$.

Так же просто преобразовать шестнадцатиричное число в двоичное. Из табл. 4.5 получаем:

$$3FA_{(16)} = 0011\ 1111\ 1010_{(2)}.$$

Простота взаимного преобразования двоичных и шестнадцатиричных чисел и является причиной использования шестнадцатиричной системы счисления для сокращенной записи двоичных чисел.

4.10. Двоично-десятичный код

Двоично-десятичный код¹ позволяет представить в вычислительной машине десятичные цифры 0, ..., 9 с помощью символов двоичного алфавита 0 и 1 (табл. 4.6).

Т а б л и ц а 4.6

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

При двоично-десятичном кодировании каждая десятичная цифра заменяется соответствующим 4-разрядным двоичным числом. При этом, хотя с помощью 4 бит можно представить десятичные числа в диапазоне от 0 до 15, числа от 10 до 15 специально не кодируются. Например, при двоично-десятичном кодировании числа 13 цифры 1 и 3 кодируются в отдельности, т. е. код числа 13 имеет вид: 0001 0011.

Выводы

1. Один байт состоит из 2 тетрад или 8 бит.
2. Шестнадцатиричная система счисления позволяет сокращенно представлять двоичные числа.
3. Десятичные цифры 0, ..., 9 можно закодировать в двоичном виде с помощью двоично-десятичного кода.

4.11. Системы кодирования

Как известно, вычислительная машина непосредственно способна оперировать лишь нулями и единицами, в то время как в нашей повседневной жизни для записи информации используется широкий набор различных букв, цифр и других символов. Например, располагая буквы алфавита в нужной последовательности, строят предложение. С помощью цифровых символов 0, ..., 9 можно записать любое число. Наконец, часто приходится использовать символы ., ? ; % и т. п. Чтобы записанную подобным образом информацию можно было ввести в цифровую машину или систему, необходимо разработать специальный

¹ В оригинале BCD — аббревиатура от слов Binary Coded Decimal. (Прим. ред.)

Таблица 4.7

Шестнадцатичное представление числа	ASCII (7 бит)	EBCDIC (8 бит)	Шестнадцатичное представление числа	ASCII (7 бит)	EBCDIC (8 бит)	Шестнадцатичное представление числа	ASCII (7 бит)	EBCDIC (8 бит)
0			2A	*		54	T	
1			2B	+		55	U	
2			2C			56	V	
3			2D			57	W	
4			2E			58	X	
5			2F	/		59	Y	
6			30	0		5A	Z	
7			31	1		5B		\$
8			32	2		5C		*
9			33	3		5D)
A			34	4		5E		
B			35	5		5F		
C			36	6		60		
D			37	7		61	a	
E			38	8		62	b	
F			39	9		63	c	
10			3A			64	d	
11			3B			65	e	
12			3C	<		66	f	
13			3D	=		67	g	
14			3E	>		68	h	
15			3F	?		69	i	
16			40		Пробел	6A	j	
17			41	A		6B	k	
18			42	B		6C	l	%
19			43	C		6D	m	
A			44	D		6E	n	
B			45	E		6F	o	>
C			46	F		70	p	?
D			47	G		71	q	
E			48	H		72	r	
F			49	I		73	s	
20	Пробел		4A	J		74	t	
21	!		4B	K		75	u	
22	"		4C	L		76	v	
23	#		4D	M		77	w	
24	\$		4E	N		78	x	
25	%		4F	O		79	y	
26	&		50	P		7A	z	
27			51	Q		7B		
28	(52	R		7C		≠
29)		53	S		7D		

Шестнадцатичное представление числа	ASCII (7 бит)	EBCDIC (8 бит)	Шестнадцатичное представление числа	ASCII (7 бит)	EBCDIC (8 бит)	Шестнадцатичное представление числа	ASCII (7 бит)
7E		=	A9		z	D4	
7F		"	AA			D5	
80			AB			D6	
81		a	AC			D7	
82		b	AD			D8	
83		c	AE			D9	
84		d	AF			DA	
85		e	B0			DB	
86		f	B1			DC	
87		g	B2			DD	
88		h	B3			DE	
89		i	B4			EF	
8A			B5			E0	
8B			B6			E1	
8C			B7			E2	
8D			B8			E3	
8E			B9			E4	
8F			BA			E5	
90			BB			E6	
91		j	BC			E7	
92		k	BD			E8	
93		l	BE			E9	
94		m	BF			EA	
95		n	C0		A	EB	
96		o	C1		B	EC	
97		p	C2		C	ED	
98		q	C3		D	EE	
99		r	C4		E	F0	
9A			C5		F	F1	
9B			C6		G	F2	
9C			C7		H	F3	
9D			C8		I	F4	
9E			C9			F5	
9F			CA			F6	
A0			CB			F7	
A1			CC			F8	
A2		s	CD			F9	
A3		t	CE			FA	
A4		u	CF			FB	
A5		v	D0		J	FC	
A6		w	D1		K	FD	
A7		x	D2		L	FE	
A8		y	D3			FF	

метод кодирования, который ставил бы в соответствие каждому исходному символу определенную комбинацию нулей и единиц.

В микро-ЭВМ наиболее часто употребляется два кода: расширенный двончно-кодированный EBCDIC (Extended Binary Coded Decimal Interchange) и американский стандартный для обмена информацией ASCII (American Standard Code for Information Interchange). В табл. 4.7 перечислены восьмиразрядные комбинации нулей и единиц вместе с их шестнадцатиричными эквивалентами, которые ставятся в соответствие алфавитно-цифровым символам в кодах ASCII и EBCDIC.

Заметим, что в коде ASCII для представления информации используется в действительности не восемь, а семь разрядов, и поэтому содержание самого левого из восьми разрядов роли не играет. Например, для буквы «а» в таблице содержится код $61_{(16)} = 01100001_{(2)}$, который можно записать как $100001_{(2)}$.

Глава пятая

СХЕМОТЕХНИКА ЭВМ

5.1. Введение

В этой главе рассмотрены важнейшие вопросы схемотехники ЭВМ. Излагаемый материал может оказаться в большей степени знакомым читателю, которому приходилось заниматься аппаратными средствами ЭВМ, и в меньшей степени специалисту по программированию.

Центральный процессор является лишь одной из многих интегральных микросхем, входящих в состав микро-ЭВМ, как это видно, например, из рис. 2.1. Цель настоящей главы заключается в том, чтобы, во-первых, познакомить читателя с назначением элементов, не входящих в состав центрального процессора, и, во-вторых, рассмотреть схемотехнику ЦП, что позволит понять, какую роль играют соответствующие схемы в выполнении команд микро-ЭВМ.

Первая категория схем включает в себя логические элементы типа И-НЕ, ИЛИ-НЕ, инверторы, а также триггеры. Ко второй категории относятся схемы, реализующие логические функции И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ, а также регистры и счетчики.

Структура микро-ЭВМ изображена на рис. 2.4 (см. также рис. 2.1), из которого видно, из каких узлов состоит микро-ЭВМ и как эти узлы соединены с шинами данных и адресной. Микропроцессор 8080А и генератор тактовых сигналов 8224 рассматриваются в гл. 8. В гл. 6 рассматривается основная память, которая состоит из ПЗУ (четыре схемы типа 464) и ОЗУ (восемь схем типа 5101). В гл. 20 анализируется устройство ввода-вывода, представленное на рис. 2.4 в виде схемы 8255. Дисплей и панель управления с шестнадцатиричной клавиатурой изображены в правой нижней части рис. 2.4. Подробно они рассмотрены в гл. 16.

Все схемы, рассматриваемые в данной главе, являются цифровыми. На их входах и выходах могут появляться лишь два характерных уровня напряжения: низкое (около 0 В) и высокое (например, 5 В), причем низкий и высокий уровни напряжения представляют собой двоичные значения 0 и 1 соответственно.

5.2. Логический элемент И

Условное графическое изображение логического элемента И, представленное на рис. 5.1, *а* соответствует американской спецификации MIL. Обозначение того же элемента на рис. 5.1, *б* отвечает европейскому стандарту. На этих рисунках изображен логический элемент И с тремя входами: *A*, *B* и *C*. Выход схемы обозначен буквой *F*. Напряжение подается на вход между каждым из зажимов *A*, *B*, *C* и общей точкой

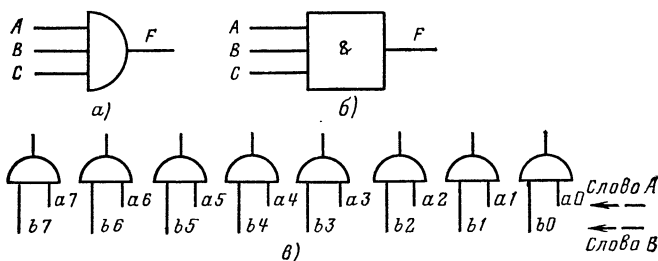


Рис. 5.1.

(«землей»), причем, так как речь идет о цифровой схеме, на входе может появиться напряжение лишь одного из двух уровней: 0 или 5 В. С учетом этого можно образовать восемь комбинаций напряжений на входах *A*, *B* и *C* (табл. 5.1).

Таблица 5.1. Комбинации сигналов на входах элемента И

$U_A, В$	$U_B, В$	$U_C, В$
0	0	0
0	0	5
0	5	0
0	5	5
5	0	0
5	0	5
5	5	0
5	5	5

Таблица 5.2. Таблица истинности логического элемента И

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Если считать, что двоичным значениям 0 и 1 соответствуют уровни 0 и 5 В, то на основе табл. 5.1 может быть построена так называемая таблица истинности, в которой приводятся значения выходной переменной для всех возможных сочетаний входных переменных (табл. 5.2). Работа логического элемента И описывается следующим правилом: выходной сигнал равен 1 только тогда, когда все входные сигналы равны 1, т. е. $F = 1$ только тогда, когда $A = 1$, $B = 1$ и $C = 1$. Если же по крайней мере на одном из входов появляется 0, то и на выходе будет 0.

Это правило работы логического элемента И аналитически записывается в виде

$$F = A \cdot B \cdot C, \quad (5.1)$$

т. е. операция И изображается точкой.

Подобная запись заимствована из алгебры Буля, которую можно рассматривать как арифметику систем, построенных на логических схемах.

Арифметическо-логическое устройство микро-ЭВМ, как это изображено на рис. 5.1, *в*, содержит восемь двухвходовых логических элементов И, что позволяет микро-ЭВМ выполнить операцию И над двумя словами А и В, каждое длиной 8 бит. При выполнении АЛУ операции И над словами 110110110 и 01001111 результатом будет 00000110, так как лишь во втором и третьем разрядах обоих слов стоят единицы.

5.3. Логический элемент ИЛИ

На рис. 5.2, *а* представлено условное графическое обозначение логического элемента ИЛИ, соответствующее спецификации MIL. Обозначение того же элемента, отвечающее принятому в настоящее время европейскому стандарту, приведено на рис. 5.2, *б*. Изображение соответствует логическому элементу ИЛИ с тремя входами.

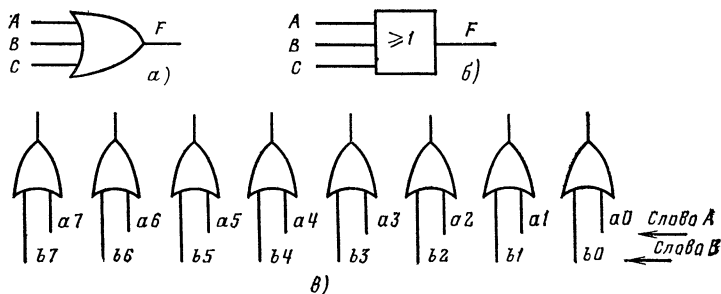


Рис. 5.2.

Выходной сигнал логического элемента ИЛИ равен единице, если хотя бы один из входных сигналов равен единице, т. е. $F = 1$, если $A = 1$ и (или) $B = 1$ и (или) $C = 1$ (табл. 5.3). Аналитически это правило записывается следующим образом:

$$F = A + B + C. \quad (5.2)$$

Символ «+» обозначает выполнение операции ИЛИ. Такое использование этого символа, заимствованное из алгебры Буля, имеет совершенно иной смысл, чем в обычной алгебре.

Арифметическо-логическое устройство микро-ЭВМ (см. рис. 5.2, *в*), содержащее восемь двухвходовых логических элементов ИЛИ, способно выполнить операцию ИЛИ над двумя 8-битными словами. Если микро-ЭВМ обрабатывает, например, слова 10010110 и 11100001, то результатом выполнения над ними операции ИЛИ будет 11110111, так как лишь в четвертом разряде обоих слов стоит ноль.

Таблица 5.3. Таблица истинности логического элемента ИЛИ

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Таблица 5.4. Таблица истинности логического элемента И-НЕ

A	B	C	D	F
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

Выводы

1. Выходной сигнал логического элемента И равен единице, если все входные сигналы равны единице.
2. Выходной сигнал логического элемента ИЛИ равен единице, если по крайней мере один из входных сигналов равен единице.

5.4. Инвертор

Выходной сигнал инвертора противоположен по значению входному сигналу или, другими словами, сигналы на входе и на выходе инвертора всегда не совпадают: если на входе 0, то на выходе 1; если на входе 1, то на выходе 0.

На рис. 5.3, а дано условное графическое изображение инвертора, соответствующее спецификации МІЛ. Кружок на выходе означает выполнение операции НЕ. Условное изображение инвертора по европейскому стандарту показано на рис. 5.3, б.

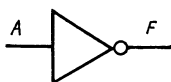
Так как выходной сигнал инвертора противоположен по значению входному, он также называется логическим элементом НЕ.

Операция отрицания аналитически записывается в виде

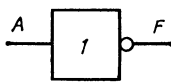
$$F = \bar{A} \quad (5.3)$$

Горизонтальная черта над переменной А читается как «не», т. е. $F = \bar{A}$ читается как «F равно не А».

На рис. 5.4 показана функциональная логическая схема, составленная из рассмотренных элементов. Цифрой 1 обозначен логический элемент И, выходной сигнал которого равен нулю. На выходе В инвертора 3 сигнал равен единице. Логический элемент 2 выполняет операцию ИЛИ, его выходной сигнал равен единице. Элемент 4 — это тоже инвертор, сигнал которого на вы-



а)



б)

Рис. 5.3.

ходе D равен нулю, причем, так как сигнал на входе В логического элемента 5 равен единице, на его выходе в результате выполнения операции ИЛИ появляется сигнал E, равный единице.

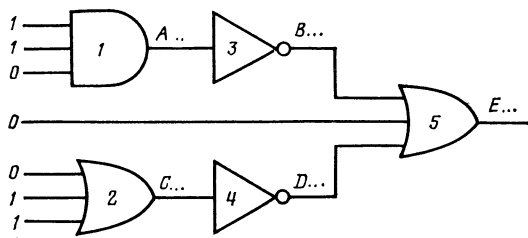


Рис. 5.4.

Как видно из рис. 5.4, инверторы широко используются для построения микро-ЭВМ. Например, в АЛУ для нахождения дополнительного кода числа приходится инвертировать значения всех его бит.

5.5. Логический элемент И-НЕ

Как видно из табл. 5.4, для схемы, изображенной на рис. 5.5, сигнал D принимает значение 1, если все его входные сигналы равны 1. Поэтому столбец D содержит единицу лишь в последней строке таблицы.

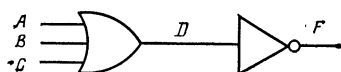


Рис. 5.5.

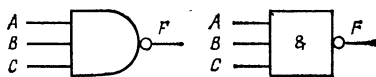


Рис. 5.6.

В результате инвертирования сигнал F противоположен по значению сигналу D, вследствие чего лишь в последней строке столбца F стоит 0. Таким образом, F равно нулю только тогда, когда все входные сигналы равны единице.

Такая комбинация логических элементов И и НЕ образует так называемый логический элемент И-НЕ. Подобные логические элементы встречаются весьма часто в силу того, что любая логическая схема может быть построена исключительно на элементах И-НЕ, т. е. в принципе можно всегда обойтись элементом одного единственного типа.

Правило работы логического элемента И-НЕ можно сформулировать так: выходной сигнал равен нулю, если все входные сигналы равны единице.

Условное графическое обозначение логического элемента И-НЕ такое же, как и для элемента И, с добавлением кружка на выходе (рис. 5.6). В соответствии с алгеброй Буля операция И-НЕ аналитиче-

ски записывается следующим образом:

$$F = \overline{A \cdot B \cdot C}. \quad (5.4)$$

Над переменными А, В и С сначала совершается операция И, результат которой инвертируется.

В отличие от логических элементов И, ИЛИ и НЕ элементы И-НЕ в АЛУ не используются. Тем не менее в микро-ЭВМ существует возможность выполнения операции И-НЕ над двумя словами. Для этого над словами сначала выполняется операция И, все биты результата которой инвертируются.

Выводы

1. Выходной сигнал логического элемента НЕ противоположен по значению входному сигналу.

2. Результат выполнения операции НЕ над переменными А, В и F записывается как А, В и \bar{F} .

3. Выходной сигнал логического элемента И-НЕ равен нулю, если все входные сигналы равны единице.

5.6. Логический элемент ИЛИ-НЕ

Из табл. 5.5 для схемы, изображенной на рис. 5.7, видно, что 0 стоит только в первой строке столбца D. Поэтому, так как этот логи-

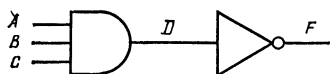
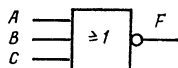


Рис. 5.7.



Рис. 5.8.



ческий элемент выполняет и операцию инвертирования, 1 появляется лишь в первой строке столбца F.

Т а б л и ц а 5.5. Таблица истинности логического элемента ИЛИ-НЕ

A	B	C	D	F	A	B	C	D	F
0	0	0	0	1	1	0	0	1	0
0	0	1	1	0	1	0	1	1	0
0	1	0	1	0	1	1	0	1	0
0	1	1	1	0	1	1	1	1	0

В логическом элементе ИЛИ-НЕ совмещено выполнение двух операций: ИЛИ и НЕ. При этом в соответствии с названием элемента сначала выполняется операция ИЛИ, а затем производится инвертирование.

В логическом элементе ИЛИ-НЕ выходной сигнал равен единице, если все входные сигналы равны нулю. В алгебре Буля операция

ИЛИ-НЕ записывается следующим образом:

$$F = \overline{A + B + C} \quad (5.5)$$

Над переменными А, В и С сначала выполняется операция ИЛИ, результат которой инвертируется. Для элемента ИЛИ-НЕ используется такое же условное графическое обозначение, как и для элемента ИЛИ, с добавлением кружка на выходе (рис. 5.8).

Так же, как и для логического элемента И-НЕ, любую логическую схему можно построить на основе элементов только одного типа: ИЛИ-НЕ. Кроме того, как и элементы И-НЕ, элементы ИЛИ-НЕ в АЛУ отсутствуют. Для выполнения в микро-ЭВМ операции ИЛИ-НЕ над двумя словами, сначала над ними необходимо выполнить операцию ИЛИ, после чего инвертировать все биты результата.

5.7. Логический элемент ИСКЛЮЧАЮЩЕЕ ИЛИ

В логическом элементе ИСКЛЮЧАЮЩЕЕ ИЛИ (рис. 5.9) выходной сигнал равен единице только тогда, когда один из входных сигналов равен единице (табл. 5.6). Если, например, АЛУ содержит восемь двухвходовых логических элементов ИСКЛЮЧАЮЩЕЕ ИЛИ, то в нем можно выполнить эту операцию над двумя словами, каждое из которых имеет длину 8 бит.

Таблица 5.6. Таблица истинности логического элемента ИСКЛЮЧАЮЩЕЕ ИЛИ

А	В	Р
0	0	0
0	1	1
1	0	1
1	1	0

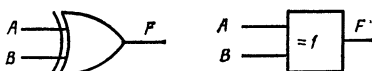


Рис. 5.9.

Пусть, например, операция ИСКЛЮЧАЮЩЕЕ ИЛИ выполняется над словами 01001111 и 10001111. Результатом будет 11000000.

Выводы

1. Логический элемент ИЛИ-НЕ дает на выходе единицу только тогда, когда на всех его входах появляются нули.
2. Логический элемент ИСКЛЮЧАЮЩЕЕ ИЛИ дает на выходе единицу, если на одном из его входов появляется 1.

5.8. Триггеры

Триггер представляет собой логическую схему с двумя выходами, значения сигналов на которых противоположны друг другу: если на одном из выходов 1, то на другом 0, и наоборот. Поэтому выходы триггера обозначают Q и \bar{Q} ; \bar{Q} по значению противоположно Q (\bar{Q} читается как «не Q»).

Триггер имеет также дополнительные входы (не менее двух): установки, обозначаемый как S (set — установка), и сброса, обозначаемый

R (reset — сброс). Появление импульса на входе S переводит триггер в состояние $Q = 1$. Такой процесс называют установкой триггера в 1. Таким образом, вход, на который нужно подать сигнал для получения на выходе Q значения 1, и называют входом установки триггера в 1. В зависимости от схемных особенностей триггера установка в 1 будет проходить при подаче на этот вход либо сигнала 1, либо сигнала 0. Вход R используют для перевода триггера в состояние $Q = 0$. Этот процесс называют установкой триггера в 0, или сбросом. Соответствующий вход называют входом установки триггера в 0, или входом сброса.

Условное графическое обозначение триггера, работу которого рассмотрели, приведено на рис. 5.10, а. Так как кружок на одном из выходов указывает на выполнение операции НЕ, обозначения Q и \bar{Q}

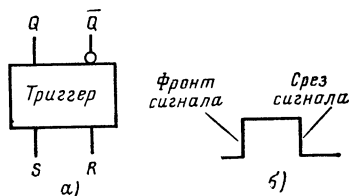


Рис. 5.10.

могут быть опущены. Часто вместо этих обозначений выходы триггера помечают символами Q_1 и Q_2 .

Важнейшим из всех возможных применений триггера является использование его в качестве запоминающего элемента, называемого защелкой. Например, ячейки статической полупроводниковой памяти строятся на триггерах, причем для запоминания в ячейке слова длиной 8 бит требуется восемь

триггеров. Другим примером применения триггера является использование его для деления частоты входных сигналов на два.

Следующие два типа триггеров наиболее важны с точки зрения их использования в микро-ЭВМ:

- а) D-триггер (применяется только как запоминающий элемент);
- б) двухтактный JK-триггер (используется как в качестве запоминающего элемента, так и для деления частоты в 2 раза).

Оба триггера имеют вход синхронизации, сигналы на который подаются от генератора тактовых сигналов. Момент времени передачи информации со входа на выход триггера определяется его типом: D-триггер реагирует на появление фронта, а JK-триггер — на появление среза синхросигнала (рис. 5.10, б).

5.8.1. D-триггер

Условное графическое изображение D-триггера представлено на рис. 5.11. Для D-триггера обязательно наличие двух входов: С (синхронизации) и D (delay — задержка). Кроме того, обычно имеются входы установки S и сброса R.

На рис. 5.12 представлена временная диаграмма работы D-триггера. Сигнал, появляющийся на входе D, передается на выход Q при переключении сигнала синхронизации из нуля в единицу. Значение сигнала на выходе Q сохраняется даже при последующих изменениях сигнала на входе D, что и позволяет использовать триггер для запоминания информации. В момент времени t_2 , когда сигнал синхронизации снова становится равным единице, новое значение сигнала со входа D передается на выход Q и запоминается.

Примечание. D-триггер, рассматриваемый в данной главе, управляется фронтом синхросигнала. Существуют и другие типы D-триггеров.

У двухтактного D-триггера на выходе повторяется то значение, которое имел входной сигнал при появлении среза синхросигнала.

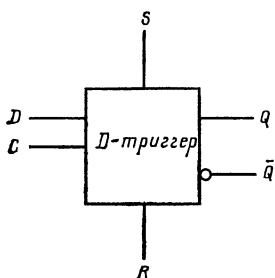
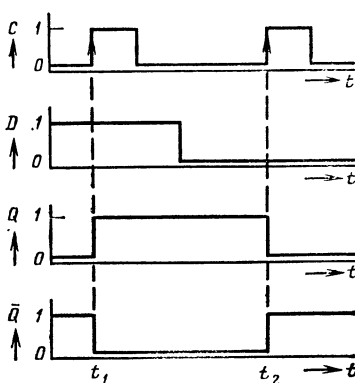


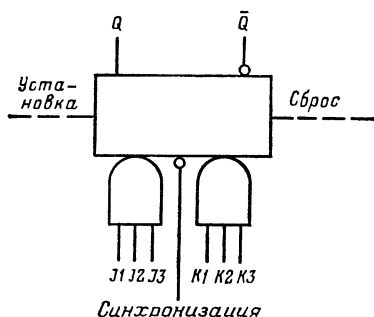
Рис. 5.11.

Рис. 5.12. →



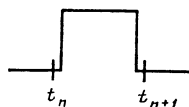
5.8.2. Двухтактный JK-триггер

На рис. 5.13 дано условное графическое изображение двухтактного JK-триггера. Триггер, изображенный на рисунке, имеет три входа J и три входа K, хотя в общем случае их может быть и другое число.



← Рис. 5.13.

Рис. 5.14.



Входы J и K предназначены для подачи на них управляющих сигналов, в то время как входы S и R используются для установки и сброса: если R становится равным нулю, то $Q = 0$, $\bar{Q} = 1$, если S становится равным нулю, то $Q = 1$, и $\bar{Q} = 0$. Триггер считается готовым к работе, если $S = 1$ и $R = 1$ (в этом случае входы S и R не оказывают влияния на его функционирование).

Входы J и K заводятся на имеющиеся в триггере логические элементы И. Таким образом, в триггере выполняются операции

$$J = J_1 \cdot J_2 \cdot J_3;$$

$$K = K_1 \cdot K_2 \cdot K_3.$$

Сигнал $J \Rightarrow 1$, если на всех входах J появляются 1. То же можно сказать и о входах K .

Момент времени, непосредственно предшествующий переключению сигнала синхронизации из нуля в единицу, обозначен t_n , а момент времени, непосредственно следующий за обратным переключением сигнала синхронизации, обозначен t_{n+1} (рис. 5.14).

Таблица 5.7 представляет собой таблицу перехода JK-триггера. Из этой таблицы можно получить целый ряд сведений о работе триггера.

Т а б л и ц а 5.7. Таблица перехода JK-триггера

Состояния входов			
в момент t_n		в момент t_{n+1}	
$J = J_1 \cdot J_2 \dots$	$K = K_1 \cdot K_2 \dots$	Q_{n+1}	\bar{Q}_{n+1}
0	0	Q_n	\bar{Q}_n
0	1	0	1
1	0	1	0
1	1	\bar{Q}_n	Q_n

а) если $J = 0$ и $K = 0$, то состояния выходов в моменты времени t_n и t_{n+1} одинаковы (первая строка);

б) если $J = 0$, то сигнал $K = 1$ передается по срезу синхросигнала на выходы и переводит их в состояния $Q = 0$, $\bar{Q} = 1$ (вторая строка);

в) если $J = 1$, то сигнал $K = 0$ передается по срезу синхросигнала на выходы и переводит их в состояния $Q = 1$, $\bar{Q} = 0$ (третья строка);

г) триггер работает как запоминающий элемент с установочными входами для комбинаций входных сигналов $J = 0$, $K = 1$ и $J = 1$, $K = 0$;

д) если $J = 1$ и $K = 1$, то состояния выходов изменяются на противоположные после окончания действия синхросигнала, т. е. $Q_{n+1} = \bar{Q}_n$; другими словами, состояния выхода Q в момент времени t_n и t_{n+1} противоположны — выход Q переключается;

е) как следует из п. «д», JK-триггер может работать как схема деления частоты входного синхросигнала на два, если на входах J и K присутствуют единичные сигналы.

Выводы

1. В D-триггере информация передается со входа D на выход Q при переключении сигнала синхронизации из нуля в единицу.

2. JK-триггер можно использовать в качестве как запоминающего элемента, так и схемы деления частоты на два.

3. JK-триггер работает как схема деления частоты следования сигналов синхронизации на два, если $J = 1$ и $K = 1$.

4. Сброс и установка JK-триггера происходит при появлении сигналов $R = 0$ и $S = 0$ соответственно.

5.9. Регистры

В предыдущем параграфе говорилось о том, что триггеры могут использоваться для запоминания информации. Так как триггер может находиться в двух состояниях, он способен хранить двоичные значения 0 и 1. Если же требуется запомнить двоичное число, составленное из некоторого набора нулей и единиц, то потребуется и соответствующее

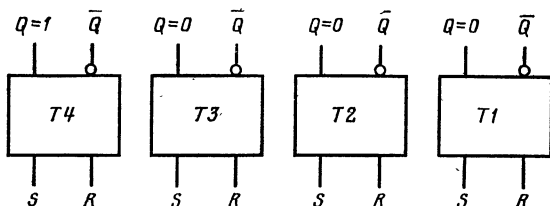


Рис. 5.15.

число триггеров. Например, для запоминания 8-разрядного числа нужно восемь триггеров: по одному триггеру на каждый бит. Совокупность триггеров, предназначенную для хранения двоичного числа определенной длины, называют *регистром*. На рис. 5.15 представлен регистр, способный хранить 4-битные слова, причем триггер Т4 используется для запоминания разряда с наибольшим весом, например при запоминании числа $1000_{(2)}$ триггер Т4 перейдет в состояние 1. Так как данный регистр может хранить слова с длиной не более 4 бит, в нем

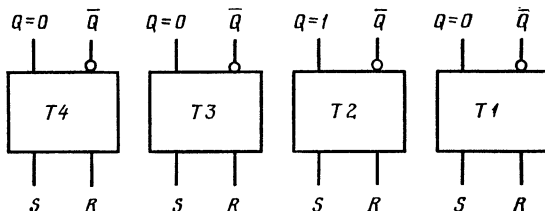


Рис. 5.16.

могут быть записаны десятичные числа от 0 до 15, двоичное представление которых требует как раз не более четырех разрядов.

Рассмотрев состояния триггеров регистра, а точнее, их выходы Q, можно определить, какое в нем хранится двоичное число. Как видно из рис. 5.16, выходной сигнал Q триггера Т4 равен нулю; для триггера Т3 $Q = 0$; для триггера Т2 $Q = 1$ и для триггера Т1 $Q = 0$. Следовательно, в регистре записано двоичное число 0010. Слева от 1 значащих цифр нет, и поэтому десятичный эквивалент числа, записанного в регистре, есть 2.

Установку триггеров регистра в состоянии 0 или 1 в соответствии со значениями разрядов входного слова называют приемом информации в регистр. Прием слова в регистр можно выполнить, подавая соответствующие сигналы на входы установки или сброса триггеров. Напри-

мер, для приема в регистр десятичного числа 12 выходы Q регистра следует перевести в состояния 1100. Это достигается установкой триггеров $T3$ и $T4$ в 1 и сбросом триггеров $T2$ и $T1$.

Микропроцессор в основном состоит из таких регистров, как регистр-аккумулятор, регистры общего назначения и регистр команд. В микро-ЭВМ, обрабатывающей слова длиной 8 бит, эти регистры содержат каждый по восемь триггеров, что и создает возможность хранения 8-битных слов.

Выводы

1. Регистры строятся на основе триггеров, действующих как запоминающие элементы.

2. В регистре для хранения каждого разряда двоичного числа требуется один триггер.

5.10. Компаратор

В микро-ЭВМ часто требуется сравнивать содержимое двух регистров или содержимое регистра и счетчика (рис. 5.17). Представим себе, например, технологический процесс, в котором необходимо про-

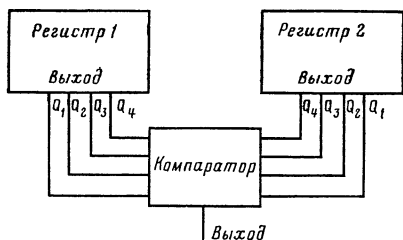


Рис. 5.17.

содержимому регистра, на упаковочный механизм посылается сигнал, прекращающий его работу. Аналогичным образом компаратор используется в АЛУ. Та команда из системы команд микро-ЭВМ, которая приводит к выполнению операции «сравнение», имеет мнемокод *CMR* (compare — сравнивать).

Содержимое двух регистров одинаково, если соответствующие триггеры в обоих регистрах находятся в одинаковых состояниях. Для сравнения содержимого этих регистров следует сравнить значения выходных сигналов всех одноименных триггеров. Если значения выходных сигналов регистра 1 совпадают со значениями выходных сигналов регистра 2, то выходным сигналом компаратора будет 1. В противном случае на выходе компаратора появляется 0.

Вывод

Содержимое двух регистров можно сравнить с помощью компаратора.

5.11. Сдвигающие регистры

По степени значимости для цифровой вычислительной техники сдвигающие регистры уступают только счетчикам. Из названия схемы очевидно, что *сдвигающий регистр* — это регистр, в котором можно производить сдвиг слова на требуемое число разрядов.

Как было сказано в § 5.9, регистр представляет собой совокупность триггеров, причем каждый триггер хранит 1 бит информации. То же справедливо и для сдвигающих регистров.

В сдвигающем регистре информация, хранящаяся в триггерах, сдвигается вправо или влево при появлении сигнала синхронизации. Заметим, что на рисунках, изображающих сдвигающие регистры, входы синхронизации не показаны. В вычислительных машинах сдвигающие регистры используются, кроме того, для выполнения операций умножения и деления.

5.11.1. Прием информации в регистр

В зависимости от способа приема информации сдвигающие регистры можно подразделить на регистры с последовательным входом и регистры с параллельным входом. В регистр, изображенный на рис. 5.18, код слова передается последовательно. В начальном состоянии все триггеры сброшены в 0. Единица сначала дается на вход триггера Т1. После появления синхросигнала 1 передается на выход триггера Т1, 0 с выхода Т1 передается на триггер Т2, а 0 с выхода триггера Т2 на триггер Т3. Далее на вход подается 0. Появляющийся вслед за этим синхросигнал передает этот 0 на выход триггера Т1. Эта единица с выхода Т1 передается на Т2 и т. д. Далее на вход подается следующая единица. Снова происходит ее передача на выход триггера Т1 и следующий бит данных сдвигается на один разряд влево. Таким образом, произошел прием слова в сдвигающий регистр путем последовательной передачи его разрядов на вход регистра по одnorазрядной шине. При этом оказалось необходимо подать на регистр три синхрои́мпульса.

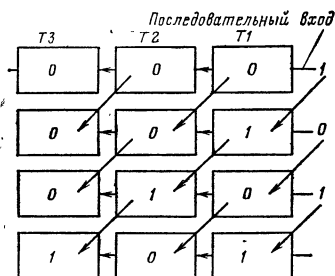


Рис. 5.18.

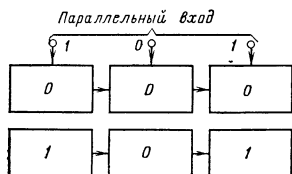


Рис. 5.19.

При параллельном способе приема информации длина слова, поступающего на вход, т. е. общее число составляющих слово «бит», должна быть равна числу параллельных входных линий, образующих шину.

5.11.2. Передача информации из регистра

На рис. 5.20 изображено, каким образом происходит передача информации из сдвигающего регистра с последовательным выходом. Передача данных при этом производится последовательно через выход триггера ТЗ. При поступлении каждого синхросигнала данные сдвигаются на один разряд влево. После поступления первого синхросигнала информация, хранившаяся в триггере Т2, передается на выход регистра. Таким образом, разряды слова, хранившегося в сдвигающем регистре, последовательно, один за другим, «выталкиваются» из регистра.

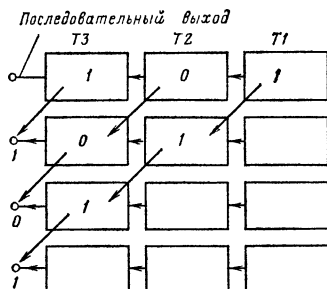


Рис. 5.20.

Существуют также сдвигающие регистры с параллельным

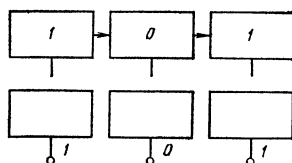


Рис. 5.21.

выходом (рис. 5.21). При появлении синхросигнала информация передается из триггеров на выходы регистра. Подобная ситуация имеет место, например, в регистре команд микропроцессора. После считывания команды из памяти она помещается в регистр команд, откуда под действием синхросигнала код команды в параллельном виде передается в дешифратор команд. Последний дешифрирует код команды и определяет, какая операция должна быть выполнена.

5.11.3. Умножение и деление

Использование сдвигающих регистров делает весьма простым процедуры деления и умножения двоичных чисел. Пусть, например, необходимо перемножить $A = 10100_{(2)}$ и $B = 10_{(2)}$. Результатом будет $C = 101000_{(2)}$:

$$\begin{array}{r} 10100 \times A \\ 10 \times B \\ \hline 101000 \quad C \end{array}$$

Если записать A в виде $0010100_{(2)}$, то очевидно, что число C можно получить, сдвинув все разряды, равные единице, на одну позицию влево. Соответственно умножение $10100_{(2)}$ на $1000_{(2)} = 8_{(10)}$ выполняется путем сдвига умножаемого на три разряда влево.

Так как деление — это операция, противоположная умножению, то оно выполняется посредством сдвига вправо. Очевидно, что, производя сдвиг слева направо или справа налево, можно делить или умножать лишь на степени числа 2. Если же необходимо умножить двоичное

число, например, на пять, то сначала необходимо сдвинуть число на два разряда влево, что эквивалентно умножению на четыре, а затем исходное число сложить с результатом сдвига. Действительно, $5 \times A$ равно $4 \times A + A$. Такой метод используется для выполнения как умножения, так и деления в аккумуляторах микро-ЭВМ. При этом содержимое аккумулятора можно сдвинуть вправо или влево с помощью специальных команд.

5.12. Специальные схемы

5.12.1. Драйвер

Драйвером называют схему, которая работает как усилитель тока. Его используют для управления лампами цифровой информации или семисегментными индикаторами. Драйверы также могут служить для соединения логических схем одной системы со схемами другой системы, если последние работают при более высоких уровнях напряжения. Наконец, драйверы часто используют при передаче цифровых данных по кабельным линиям.

5.12.2. Буферный регистр с тремя состояниями

Буферным усилителем с тремя состояниями (рис. 5.22) называют схему, предназначенную для отключения одной части микро-ЭВМ от другой. Эта схема работает подобно ключу. Если управляющий сигнал, называемый сигналом разрешения приема, равен 0, то выход переходит в высокоимпедансное состояние «не 0, не 1» (ключ отключает входы от выходов). Если же сигнал разрешения приема становится равным 1, то сигналы на выходе имеют те же значения, что и сигналы на входе. Следовательно, выход рассматриваемой схемы может находиться в одном из трех состояний: 0, 1 или состояние с высоким импедансом («плавающий потенциал»).

В большинстве случаев буферные усилители с тремя состояниями обладают чрезвычайно высокой на-

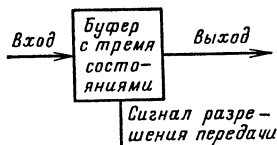


Рис. 5.22.

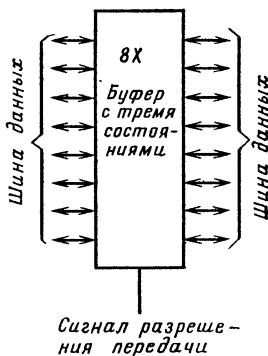


Рис. 5.23.

грузочной способностью (нагрузочная способность определяет допустимое число входов других схем, подсоединяемых к выходу данной схемы). Это позволяет сделать вывод, что подобно драйверу буферный усилитель работает как усилитель тока. Такую схему часто называют шинным драйвером.

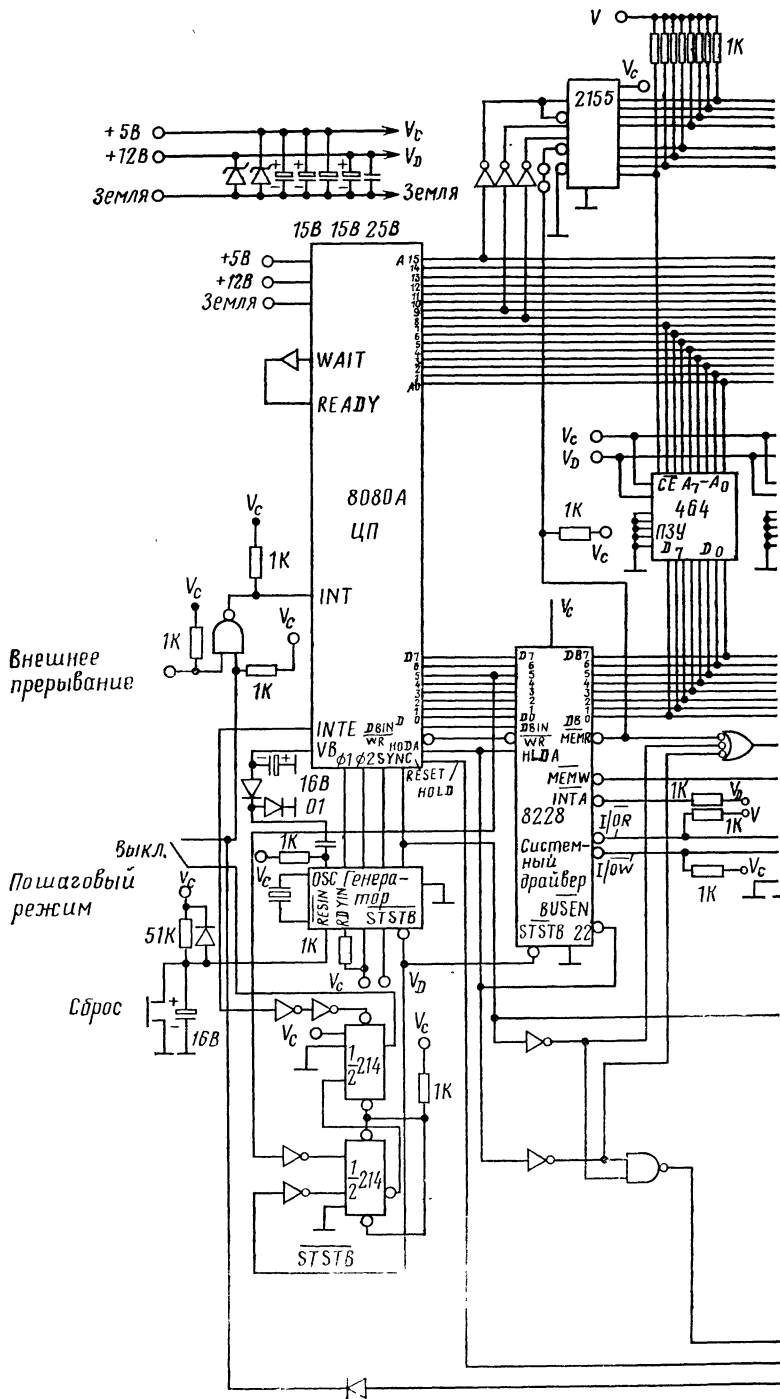
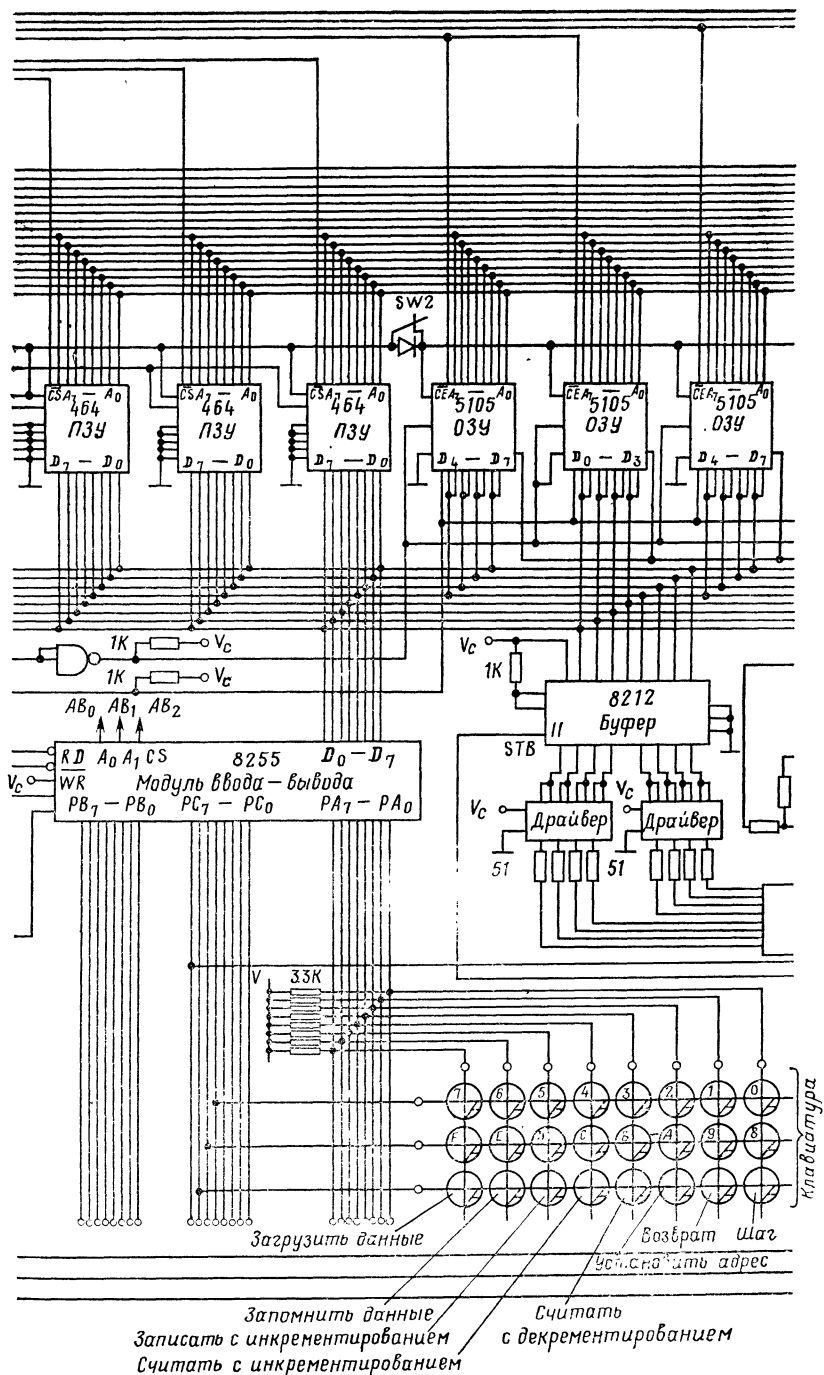


Рис. 5.24.



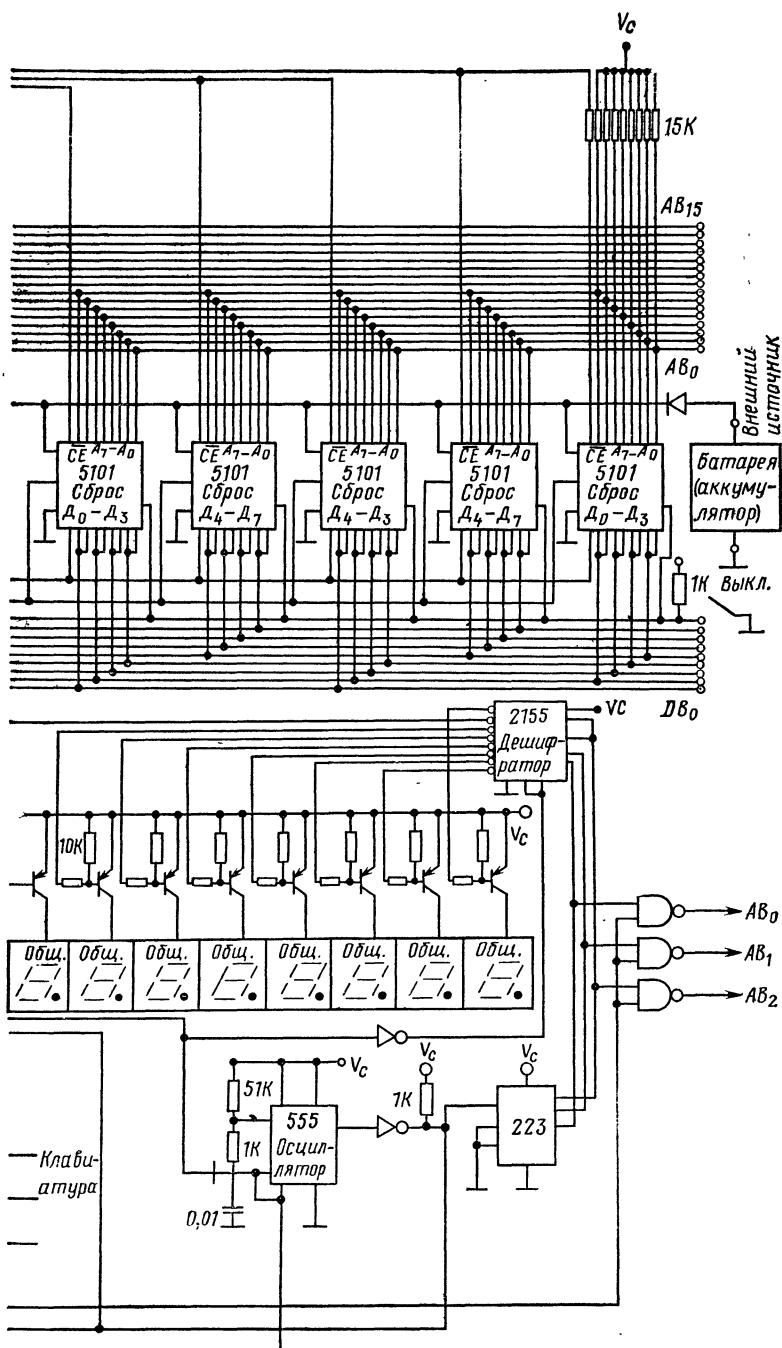


Рис. 5.24.

Буферная схема, представляющая собой восемь буферных усилителей с тремя состояниями в одном корпусе, изображена на рис. 5.23. На этом рисунке буферная схема включена в шину данных и способна передавать информацию в обоих направлениях, т. е. является двунаправленной. Шина данных на рис. 5.24 также «буферезирована» с помощью шинного драйвера на ИС типа 8228, которая способна выполнять и другие функции, не рассматриваемые в данном параграфе. На рис. 5.24 сигнал разрешения приема буфера (вывод 22) обозначен как BUSEN (BUSEnable). Интегральная микросхема типа 8212 на рис. 5.24 также представляет собой буфер. Эта схема, однако, не является двунаправленной, она служит для отключения шины данных от 7-сегментного дисплея. Для этой схемы сигнал разрешения приема (вывод 11) обозначен как STB (strobe).

Если в микро-ЭВМ имеется несколько интегральных микросхем ОЗУ или ПЗУ, то адресная шина обычно содержит один или несколько шинных драйверов (на рис. 5.24 отсутствуют).

Выводы

1. Данные, хранящиеся в сдвигающем регистре, можно сдвигать слева направо и в обратном направлении по сигналу, разрешающему сдвиг.

2. В зависимости от способа приема информации различают сдвигающие регистры с последовательным или параллельным входом.

3. Данные, хранящиеся в сдвигаемом регистре, могут передаваться из него последовательно или параллельно.

4. Сдвигающие регистры можно использовать для деления или умножения двоичных чисел.

Глава шестая

ОСНОВНАЯ ПАМЯТЬ

6.1. Введение

Ранее было показано, что последовательность команд, которые должна выполнить ЭВМ, т. е. программа, хранится в области основной памяти, называемой памятью программ. Данные, обрабатываемые по этим командам, хранятся в памяти данных.

Память программ и память данных *не обязательно* должны быть расположены отдельно. Другими словами, данные не всегда находятся в одной части памяти, а команды — в другой. Если ячейка памяти содержит команду, то эта ячейка является частью памяти программ, а если ячейка памяти содержит данные, то она является частью памяти данных вне зависимости от того, где она физически находится в ЭВМ.

Команды и данные извлекаются из памяти и передаются последовательно в ЦП во время выполнения программы.

Каждая ячейка памяти имеет свой адрес. По каждому *адресу* в памяти находится *одно*, например, 8-разрядное слово. Адрес определяет местоположение *слова*, а не *бита*. Содержимое ячейки памяти программист может интерпретировать различным образом (рис. 6.1):

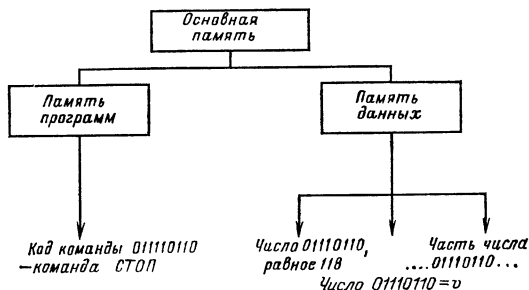


Рис. 6.1.

1. Данные, используемые для выполнения операций:
а) 8-разрядное *число*; б) *часть числа*, имеющего формат более восьми разрядов; в) число, буква или символ в соответствии с используемым кодом, таким как ASCII.

2. Команды: код операции или часть многобайтной команды.

Более подробно поясним эти вопросы в следующей главе и затем рассмотрим пример программы, который проиллюстрирует процесс взаимодействия памяти и ЦП.

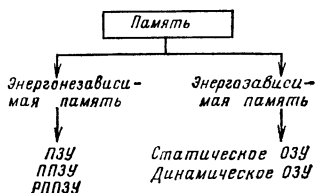
6.2. Организация памяти

Память ЭВМ состоит из множества триггерных элементов. В каждом элементе может храниться 1 бит (0 или 1). Не будем здесь рассматривать различные типы памяти (ОЗУ, ПЗУ и др.), так как они рассмотрены в гл. 3. Хотелось бы, однако, напомнить разницу между *энергозависимой* и *энергонезависимой* памятью (рис. 6.2, а).

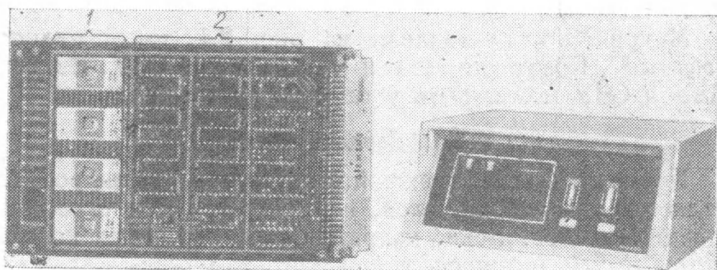
В энергозависимой памяти хранящаяся информация теряется при выключении напряжения питания. Это происходит как в статических, так и динамических ОЗУ (в динамических ОЗУ информация должна постоянно обновляться даже при включенном напряжении питания). В энергонезависимой памяти информация сохраняется при выключе-

нии напряжения питания. Примерами энергонезависимой памяти являются ПЗУ, ППЗУ и РППЗУ.

Пример *европлаты* дан на рис. 6.2, б. Европлата — это печатная плата со стандартными европейскими размерами (10×16 см). На РППЗУ видны окошки, через которые с помощью ультрафиолетового облучения стирается содержимое памяти. С помощью программирующего устройства ППЗУ, показанного на рис. 6.2, в, можно одновременно программировать



а)



б)

в)

Рис. 6.2.

два элемента РППЗУ. Для этого элементы РППЗУ снимаются с европлаты, их содержимое стирается ультрафиолетовым облучением и вставляются в гнезда ИС программирующего устройства.

6.3. Длина слова

Число различных команд, которые может выполнить ЭВМ, частично определяется разрядностью машинных слов. Большинство микро-ЭВМ имеет восемь разрядов для представления максимум 2^8 , или 256 команд. Этого обычно достаточно. В этом случае память организована таким образом, что одна ячейка памяти содержит 8 бит, и говорят о ЭВМ с длиной слова 8 бит или о *8-разрядной ЭВМ*. Слово из 8 бит называют байтом. Таким образом, формат слова в 8-разрядной ЭВМ равен 1 байту.

Некоторые микро-ЭВМ работают со словами длиной 4 или 16 бит, но они менее распространены, чем 8-разрядные

микро-ЭВМ. Длина слова в 4-разрядной ЭВМ, таким образом, равна половине байта и называется тетрадой или ниббл; 16-разрядная ЭВМ имеет длину слова 2 байта.

Емкость памяти, т. е. максимальное число машинных слов, которые могут храниться в памяти, часто выражается в килобайтах, или сокращенно Кбайт. Если память имеет емкость 1 Кбайт, то она может хранить 1024 8-разрядных слов. В вычислительной технике префикс «кило» не выражает 1000, как в метрической системе, а степень числа 2, ближайшую к 1000, которая равна $1024 = 2^{10}$. Иногда указывается число разрядов в слове и емкость памяти, например, $1 \text{ К} \times 4$ или $1 \text{ К} \times 8$. Это означает, что емкость памяти равна 1024 4-разрядных и 1024 8-разрядных слов соответственно.

Микро-ЭВМ, показанная на рис. 2.1, имеет емкость памяти $3/4$ Кбайт для ПЗУ и $1/2$ Кбайт для ОЗУ. Емкость ПЗУ и ОЗУ может быть увеличена до 1 Кбайт.

6.4. Модуль памяти

Кроме длины слова важной характеристикой микро-ЭВМ является разрядность слов, используемая для адресации машинных слов. Если, например, для адресации используется восемь разрядов, т. е. 1 байт, то микро-ЭВМ может адресовать $2^8 = 256$ различных машинных слов. Во многих

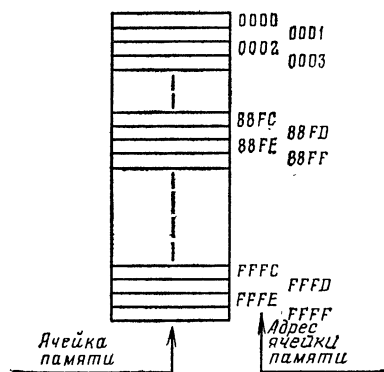


Рис. 6.3.

случаях этого недостаточно. Поэтому большинство 8-разрядных микро-ЭВМ имеют 2-байтные адреса, чтобы можно было адресовать $2^{16} = 65\,536$ слов. В этом случае самый младший адрес будет $0000000000000000_{(2)} = 0000_{(16)}$. Самый старший адрес будет $1111111111111111_{(2)} = \text{FFFF}_{(16)}$.

Эти адреса представлены в шестнадцатичных кодах на рис. 6.3.

Часто разряды одного машинного слова физически расположены в разных ИС. Интегральные микросхемы, которые вместе образуют машинное слово, называются *модулем памяти*.

Варианты модулей памяти для хранения слов разрядностью 1 байт показаны на рис. 6.4, а—в.

На рис. 6.4, а дан пример модуля памяти, состоящего из восьми ИС. Один бит каждого слова находится в одной ИС. На рис. 6.4, б показан модуль памяти, состоящий из двух ИС, в каждой из которых хранятся четыре разряда всех слов. На рис. 6.4, в дан пример размещения слова на одном

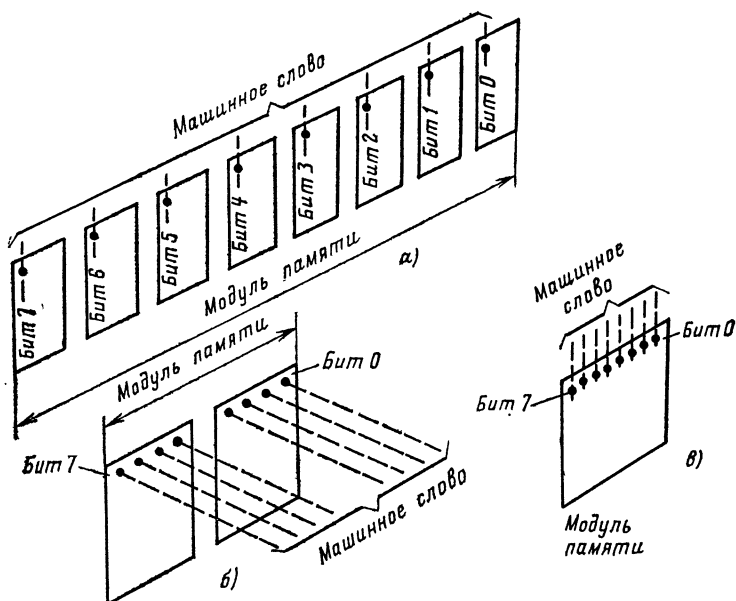


Рис. 6.4.

кристалле (ИС). Все разряды всех машинных слов размещены на одном кристалле.

Число бит, которое может храниться на одном кристалле, всегда является степенью числа 2. Часто можно встретить такие значения, как 1024, 4096, 16 384. Современная полупроводниковая память обычно организована, как показано на рис. 6.4, а, где каждая ИС содержит 1 бит каждого слова. Число машинных слов, которые могут храниться в одном модуле памяти, организованном в соответствии с рис. 6.4, а, равно числу бит, которые могут храниться на одном кристалле. Из рис. 5.24 видно, что в ПЗУ все восемь разрядов слова хранятся на одном кристалле, так как каждый кристалл имеет восемь подсоединений к шине

данных (ПЗУ типа 464). В ОЗУ, однако, только четыре разряда каждого слова хранятся на каждом кристалле; каждый кристалл имеет только четыре подсоединения к шине данных. В этом типе микро-ЭВМ каждый модуль ОЗУ всегда состоит из двух кристаллов (ОЗУ типа 5101).

Выводы

1. В отличие от энергонезависимой памяти, в энергозависимой памяти вся хранимая информация теряется всегда, когда отключается напряжение питания.

2. Длина слова микро-ЭВМ равна числу разрядов машинных слов, хранимых в памяти.

3. 8 бит = 1 байт = 2 тетрады (нибля).

4. Емкость памяти 1 Кбайт означает емкость 1024 машинных слова, разрядность каждого из которых равна 8.

Емкость $1 \text{ К} \times 4$ означает емкость 1024 4-разрядных машинных слова.

6.5. Адресация

Число разрядов в адресе памяти зависит от емкости памяти, т. е. от числа хранимых машинных слов. Если емкость памяти составляет 1 Кбайт = 1024 слов, то необходимо использовать 10-разрядный адрес ($2^{10} = 1024$).

Если основная память микро-ЭВМ содержит более одного модуля памяти, часть кода адреса должна указывать, в каком модуле памяти расположено данное слово. Эта часть называется кодом *выбора модуля* или кодом *выбора кристалла*. Выбор модуля — это более удачный термин, так как выбираем модуль, а не один кристалл. Часть кода адреса, которая выбирает слово памяти внутри модуля, называется *адресом слова*. Пример такого адреса дан на рис. 6.5.

0011	011100101101
Выбор	Адрес слова в модуле
модуля	

Рис. 6.5.

Декодирование адреса слова осуществляется в самих кристаллах памяти. Для декодирования адреса модуля (кристалла) используется отдельная ИС, как это видно из рис. 5.24 (2155).

От особенностей организации основной памяти зависит способ адресации, т. е. какие разряды адреса используются для выбора модуля и какие для адресации слова в модуле.

Примечание. Необходимо иметь в виду, что микро-ЭВМ может содержать различные типы памяти, например модуль ОЗУ (рис. 6.4, а) и модуль ПЗУ (рис. 6.4, в). При адресации в соответствии с системой выборки модуля и адресом слова необходимо быть последовательными в ее использовании, и в этом случае модули ПЗУ, ОЗУ и т. п. имеют свои коды выборки модуля.

Пример. Д а н о. Основная память микро-ЭВМ состоит из четырех модулей: одного модуля ПЗУ и трех модулей ОЗУ; емкость модуля ПЗУ составляет 1 Кбайт = 2^{10} слов;

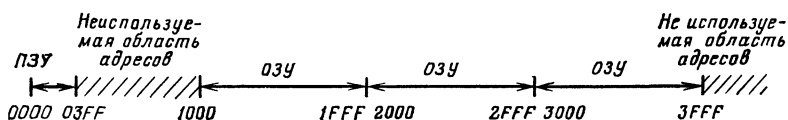


Рис. 6.6.

емкость каждого из модулей ОЗУ составляет 4 Кбайт = 2^{12} слов; адресация памяти в данной микро-ЭВМ осуществляется по 16-разрядной шине.

В о п р о с. Как адресовать слово?

В о з м о ж н о е р е ш е н и е. Чтобы выбрать слово в модуле ПЗУ, требуется десять разрядов для адреса слова ($2^{10} = 1024 = 1 \text{ К}$) и шесть разрядов для выбора модуля. Если выберем код 000000 в качестве кода выбора этого модуля ПЗУ, то адреса слов в модуле ПЗУ будут лежать в пределах от $0000000000000000_2 = 0000_{16}$ до $0000001111111111_2 = 03FF_{16}$.

Чтобы выбрать слово в одном из модулей ОЗУ, требуется 12 разрядов для адреса слова ($2^{12} = 4096 = 4 \text{ К}$). Для выбора модуля используются четыре разряда.

Если выберем код 0001 для выбора первого модуля ОЗУ, адреса слов в этом модуле будут находиться в пределах от $0001 000000000000_2 = 1000_{16}$ до $0001 111111111111_2 = 1FFF_{16}$.

Во втором модуле ОЗУ, для выбора которого используется код 0010, адреса слов будут находиться в пределах от $0010 000000000000_2 = 2000_{16}$ до $0010 111111111111_2 = 2FFF_{16}$.

В третьем модуле ОЗУ, который выбирается кодом 0011, адреса слов будут лежать в пределах от 0011 000000000000₂ = 3000₁₆ до 0011 111111111111₂ = 3FFF₁₆.

На рис. 6.6 схематически представлены адреса всех машинных слов. Такая схема называется *картой памяти*.

Выводы

1. Большинство микро-ЭВМ используют 2 байта для адресации машинных слов, т. е. может быть адресовано 65 536 слов.

2. При модульной организации памяти часть кода адреса используется для выбора модуля, а часть — для адреса слова в модуле.

3. Карта памяти — это схема, на которой адреса машинных слов поставлены в соответствие модулям памяти.

6.6. Значение машинного слова

Машинное слово в 8-разрядной микро-ЭВМ может интерпретироваться различным образом. Это могут быть: данные (8-разрядное число, часть числа разрядностью больше восьми, символ) и команды.

8-разрядное число. Когда микро-ЭВМ осуществляет вычисление, то машинное слово перемещается в АЛУ. Ряд 0 и 1 представляет в этом случае двоичное число.

В табл. 6.1 показано, каким образом микро-ЭВМ различает положительные и отрицательные числа. Отрицательные числа представлены как дополнение до числа 2

Таблица 6.1

Число					
двоичное	десятич- ное	шестнадца- тиричное	двоичное	десятич- ное	шестнадца- тиричное
10000000	—128	80	00000001	1	1
10000001	—127	81	00000010	2	2
10000010	—126	82	00000011	3	3
10000011	—125	83	⋮	⋮	⋮
⋮	⋮	⋮	01111101	+125	7D
11111110	—2	FE	01111110	+126	7E
11111111	—1	FF	01111111	+127	7F
00000000	0	0			

соответствующего положительного числа. Таким образом, представление числа 125 в двоичной системе может быть выполнено следующим образом.

Число $+125$ в двоичной системе счисления имеет код 01111101_2 . Его дополнение до числа 1 (обратный код) составляет 10000010_2 . Дополнение числа 01111101_2 до числа 2 образуется путем прибавления 1 к обратному коду числа. Представление числа -125 в двоичной системе

$$\begin{array}{r} 10000010_2 + \\ \quad \quad \quad 1_2 \\ \hline -125_{10} = 10000011_2 \end{array}$$

Из табл. 6.1 видно, что в микро-ЭВМ можно представить 8-разрядное число, которое будет находиться между -128 и $+127$. Преимущество этого вида записи заключается в том, что легко можно отличить положительное это число или отрицательное. В положительном числе старший значащий бит равен 0, в отрицательном 1.

Часть числа разрядностью больше 8. Восемьразрядным словом можно представить положительные числа от 0 до 255. Электронная вычислительная машина должна часто оперировать с большими числами, для представления которых требуется большее число разрядов. В этом случае одно число составляется из нескольких машинных слов. Используя два машинных слова (числа), можно представить любое число в диапазоне от 0 до 65535_{10} .

В принципе число слов не ограничено и может быть довольно большим. На рис. 6.7 показано 48-разрядное число, составленное из шести слов по 1 байту каждое.

11101101	10101010	11000011	00001000	11010000	00111111
Байт 5	Байт 4	Байт 3	Байт 2	Байт 1	Байт 0

48-разрядное число

Рис. 6.7.

Символ. Эффективность использования ЭВМ будет мала, если будет возможность вводить лишь данные, представленные рядами 0 и 1, или если результаты вычислений будут выводиться как аналогичные ряды единиц и нулей. Электронно-вычислительная машина должна уметь оперировать с буквами и символами, т. е. с рядами единиц и нулей, которыми закодированы буквы и символы.

Поэтому некоторая часть памяти микро-ЭВМ содержит информацию в двоичной системе, которая представляет

собой числа, в то время как другая часть используется для хранения букв или символов (*знаков*). Широко используются следующие знаки: 26 прописных букв; 26 строчных букв; около 25 символов, таких как ! ? , . / ; 10 цифр (от 0 до 9).

Чтобы избежать путаницы, которая может возникнуть из-за многообразия используемых кодов, стандартизовано несколько международных кодов для представления знаков. Наиболее часто используемыми кодами являются ASCII (американский стандартный код обмена информацией) и EBCDIC (расширенный десятичный двоично-кодированный код обмена).

Всего имеется 87 различных знаков, поэтому для их кодирования требуется 7 бит. В 8-разрядной микро-ЭВМ, которая рассматривается как базовая, остается свободным 1 бит на слово. Этот бит используется для организации системы контроля передачи данных и называется *битом паритета*. Значение бита паритета (бита контроля) (1 или 0) таково, что слово, включая контрольный бит, всегда должно содержать *четное* или *нечетное* число единиц, что оговаривается заранее. При этом говорят о «контроле по четности» или о «контроле по нечетности».

Если желаем использовать контроль по нечетности, то значение контрольного бита должно быть таким, чтобы число единиц в слове стало нечетным.

Ниже приводится несколько примеров кодов с контролем по нечетности:

10000000 — число единиц = 1
00000001 — число единиц = 1
11001011 — число единиц = 5
11011111 — число единиц = 7
01010100 — число единиц = 3

Если выбрали контроль по нечетности и встретились с словом, в котором содержится четное число единиц, знаем, что при передаче слова допущена ошибка.

В качестве примера приведем ряд кодов, составляющих слово MICRO и закодированных в ASCII (используется контроль по четности)

М	И	С	Р	О
01001101	11001001	11000011	11010010	11001111

Код команды. Ранее машинные слова интерпретировались как данные (числа и знаки). Машинное слово может быть также новой командой для ЭВМ или частью многобайтовой команды.

Выводы

1. Машинные слова можно интерпретировать как: 1) данные (8-разрядное число, часть числа большой разрядности, символ); 2) команды.

2. Отрицательное число может быть представлено в микро-ЭВМ как дополнение до числа 2 соответствующего положительного числа.

3. Число может быть представлено любым количеством машинных слов.

4. Знак — это буква, цифра или символ, закодированные в стандартном коде.

5. Когда машинное слово представляет собой однобайтную команду, то речь идет о ходе операции.

6.7. Пример

Рассмотрим работу микро-ЭВМ в процессе выполнения команды. В качестве примера возьмем команду ADD r, по которой содержимое регистра r (одного из регистров об-

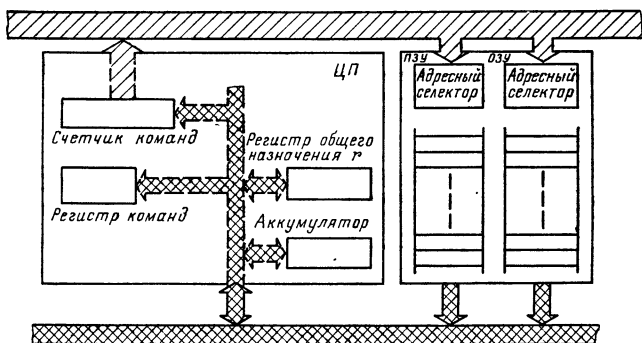


Рис. 6.8.

щего назначения) должно суммироваться с содержимым аккумулятора и результат операции должен фиксироваться в аккумуляторе.

Цикл команды в ЭВМ состоит из двух фаз: *выборки и исполнения* команды. В первой фазе команда вызывается из памяти и счетчик команд инкрементируется. Во второй фазе команда исполняется в ЦП.

Для управления работой ЦП в цикле команды используются два регистра. Одним из них является счетчик команд,

который обеспечивает последовательную выборку команд. Другим регистром является регистр команд, который

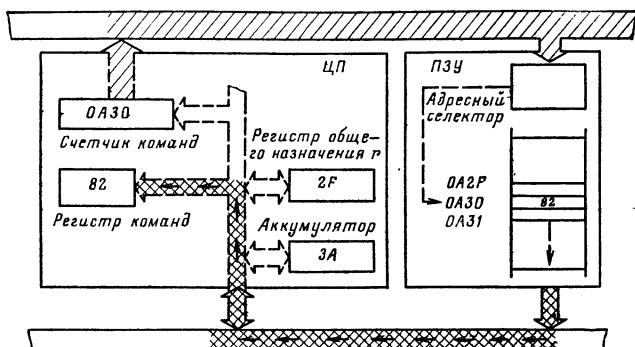


Рис. 6.9.

используется для временного хранения команды, выбранной из памяти программ. Блок управления дешифрирует содержимое регистра команд для определения операции, которую необходимо выполнить, и операндов. В рассматриваемом примере операнды хранятся в регистре общего назначения r и в аккумуляторе (рис. 6.8).

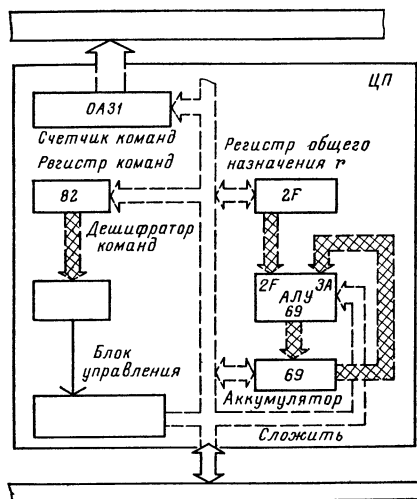


Рис. 6.10.

Сделаем следующее предположение (рис. 6.9):

а) счетчик команд находится в состоянии $0A30_{16}$; другими словами, команда, подлежащая выполнению, расположена в ячейке памяти с адресом $0A30_{16}$;

б) содержимое машинного слова с адре-

сом $AA30_{16}$ есть $10000010_2 = 82_{16}$; это код команды ADD r ;

в) содержимое регистра общего назначения r и аккумулятора до выполнения команды соответственно равно $2F_{16}$ и $3A_{16}$.

Выборка команды. Выбирается содержимое ячейки памяти, адрес которой зафиксирован в счетчике команд. В рассматриваемом случае по адресу $0A30_{16}$ содержится код команды, который загружается в регистр команд. Это положение представлено на рис. 6.9. Содержимое счетчика команд теперь инкрементируется, чтобы подготовить ЭВМ к выборке следующей команды после выполнения данной команды. Содержимое счетчика команд становится равным $0A31_{16}$. На этом выборка команды завершается.

Выполнение команды. Микро-ЭВМ дешифрирует содержимое регистра команд и определяет адреса операндов. В соответствии с кодом операции в АЛУ выполняется суммирование машинных слов и сумма фиксируется в аккумуляторе (рис. 6.10).

6.8. Цикл управления фон-Неймана

Выборка команды и выполнение команды часто описываются с помощью так называемого *цикла управления фон-Неймана*, который показан на рис. 6.11. В фазе выборки

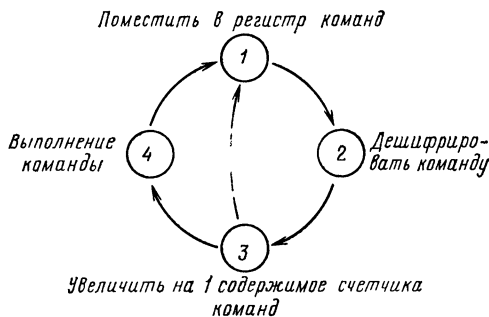


Рис. 6.11.

команды 1 содержимое ячейки памяти, адресуемой счетчиком команд, выбирается из памяти и помещается в регистр команд. Код команды поступает с регистра команд в дешифратор команд 2, где он преобразуется в систему управляющих сигналов. После этого счетчик команд инкрементируется 3. Если команда состоит из нескольких байт, то

процесс 1—3 повторяется. Вернемся к этому в следующей главе. Если команда содержит только 1 байт, то можно немедленно приступить к фазе выполнения команды 4. Когда команда выполнена, возвращаемся к 1 и осуществляем выборку следующей команды.

П р и м е ч а н и е. 1—3 вместе образуют фазу выборки команды. Когда команда состоит из 2 или 3 байт, фаза выборки команды повторяется соответственно 1 или 2 раза.

Выводы

1. Микро-ЭВМ выполняет команду в две фазы: 1) выборка команды, во время которой команда выбирается из памяти и помещается в регистр команд; 2) исполнение команды, во время которой выполняется требуемая операция.

2. Программист должен определить и помнить, где в памяти находятся числа, знаки и коды команд. Что касается микро-ЭВМ, то для нее содержимое ячейки памяти не что иное, как набор 0 и 1.

Глава седьмая

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

7.1. Введение

В настоящей главе проиллюстрируем на примере простой программы, выполняющей сложение, каким образом ЭВМ обрабатывает информацию.

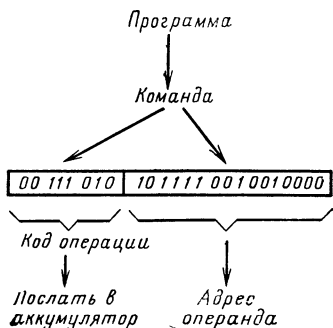


Рис. 7.1.

Кроме того, покажем, как программа и данные вводятся в машину. Данными могут быть цифры, буквы, знаки препинания, световые сигналы, звуковые сигналы и т. п. Программа (рис. 7.1) состоит из команд. Команда содержит код операции, который сообщает ЭВМ, что нужно сделать. За кодом операции идет указание о том, где находит-

ся объект действия. Примером кода операции может служить операция ПОСЛАТЬ В АККУМУЛЯТОР. За кодом операции следует указание о том, где располагаются обрабатываемые данные. В примере, приведенном на рис. 7.1, информация расположена в ячейке памяти.

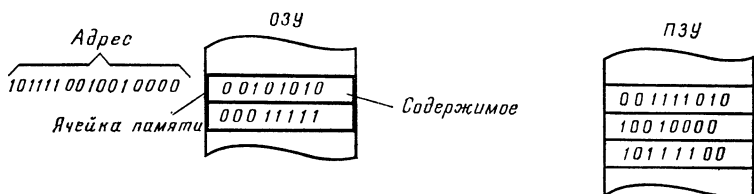


Рис. 7.2.

Программа и данные хранятся в основной памяти (рис. 7.2). Программа размещается в ПЗУ, а данные — в ОЗУ. Память состоит из *ячеек*. Ячейка имеет *адрес* и *содержимое*. Указав адрес ячейки, получим доступ к ее содержимому.

7.2. Шестнадцатиричные числа

На рис. 7.1. и 7.2 команды, содержимое ячеек памяти и адреса ячеек памяти закодированы комбинациями 0 и 1. Такой способ записи называется *машинным языком*. Как известно, ЭВМ построена на основе схем, на выходе которых появляется или не появляется сигнал. Наличие или отсутствие сигнала обозначаем 1 и 0 соответственно. В этом смысле 1 и 0 не являются цифрами, а служат для обозначения некоторых состояний электронной схемы (табл. 7.1).

Таблица 7.1

Число		Число		Число	
двоичное	шестнадцатиричное	двоичное	шестнадцатиричное	двоичное	шестнадцатиричное
0000	0	0101	5	1010	A
0001	1	0110	6	1011	B
0010	2	0111	7	1100	C
0011	3	1000	8	1101	D
0100	4	1001	9	1110	E
				1111	F

Когда называют или записывают команды и адреса и содержимое ячеек памяти с помощью последовательностей 0 и 1, это не только отнимает много времени, но также создает возможность появления ошибок. По этой причине применяют более компактную систему счисления для обозначения двоичных чисел, используемых в ЭВМ. Такая система обозначений называется шестнадцатиричной системой счисления. Шестнадцатиричная система счисления использует 16 символов: десять цифровых (0—9) и шесть буквенных (A — F). Каждый из этих символов соответствует определенной комбинации из 4 бит, как это показано в табл. 7.1. Чтобы преобразовать двоичное число в шестнадцатиричное, нужно разбить его на группы, содержащие по 4 бита. Если длина двоичного числа не кратна четырем, то число дополняется слева нулями так, чтобы его длина стала кратной четырем. Если перекодировать адрес, показанный на рис. 7.2, в шестнадцатиричную систему счисления, то получится BC90. Содержимое ОЗУ равно 2A и 3F.

Поскольку ячейка памяти имеет длину 8 бит, можно закодировать ее содержимое двумя шестнадцатиричными цифрами. Если адрес памяти имеет длину 16 бит, то для его представления понадобятся четыре шестнадцатиричные цифры. Если адрес памяти имеет меньшую длину, то, естественно, для его представления требуется меньшее число шестнадцатиричных цифр.

7.3. Универсальная программа

Когда пишут программу, она должна быть универсальной. Если пишут программу, выполняющую сложение, то необходимо написать ее так, чтобы она была пригодна для сложения любых чисел. Это означает, что нельзя включать числа в саму программу, поскольку они в разных случаях будут различными. Эту проблему решают указанием в программе того места, где находятся интересующие нас числа. Допустим, что в программе сложения два суммируемых

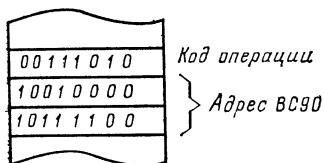


Рис. 7.3.

числа расположены в памяти по адресам BC90 и BC91, а сумма по адресу BC92. Если теперь необходимо над первым из суммируемых чисел выполнить операцию, то в коде

операции для соответствующей команды указывают, например, операцию ПОСЛАТЬ В АККУМУЛЯТОР. При этом указывают, что число находится по адресу ВС90.

В данном случае команда имеет длину 3 байта. Один байт занимает код операции и 2 байта адрес. Полная длина команды составляет 3 байта и говорят, что это *3-байтная команда*. Эта команда располагается в ОЗУ, как это показано на рис. 7.3. Первая ячейка памяти содержит код операции. Вторая ячейка памяти содержит младшие 8 бит адреса памяти, по которому расположено первое слагаемое. Третья ячейка содержит старшие 8 бит этого адреса.

Выводы

1. Команда состоит из кода операции, который показывает, какое действие должна выполнить микро-ЭВМ, и адресной части, которая показывает, где (или над каким объектом) это действие должно быть выполнено.

2. Электронная вычислительная машина построена на основе схем с двумя устойчивыми состояниями, на выходе которых присутствует или отсутствует потенциальный сигнал. Эти состояния кодируются 1 и 0.

3. При обозначении адреса или содержимого памяти и для записи команд используется шестнадцатиричная система счисления. Группа из 4 бит кодируется одной шестнадцатиричной цифрой.

4. Чтобы сделать программу пригодной для разных случаев, указывают в команде не сами данные, а адреса ячеек памяти, в которых они расположены.

5. Команда, с помощью которой обрабатываются данные, хранимые в памяти, имеет длину 3 байта: 1 байт служит для указания кода операции и 2 байта для задания адреса ячейки памяти.

7.4. Выполнение программы суммирования

На рис. 7.4 изображен процесс выполнения в микро-ЭВМ программы, осуществляющей сложение двух чисел. Суммируемые числа размещены в памяти данных (в ОЗУ) по адресам ВС90 и ВС91. Сумма помещается по адресу ВС92. Эту информацию будем использовать при записи команд. Предположим, что суммируемые числа имеют значения 2A и 3F.

Команды программы последовательно располагаются в памяти программ (в ПЗУ). В рассматриваемом примере суммирование может быть выполнено с помощью трех команд. Каждая команда состоит из кода операции, за которым следует адрес ячейки памяти, откуда данные могут быть взяты или где они могут быть размещены. Обработка происходит в центральном процессоре. Операцию сложения выполняет АЛУ. Прежде чем выполнить сложение, следует

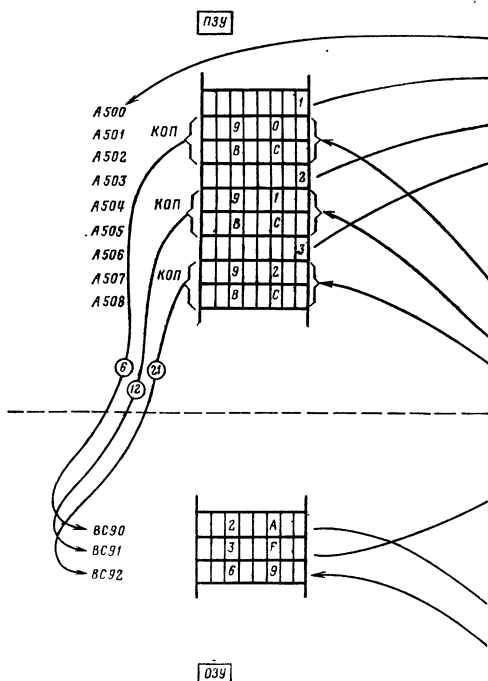


Рис. 7.4.

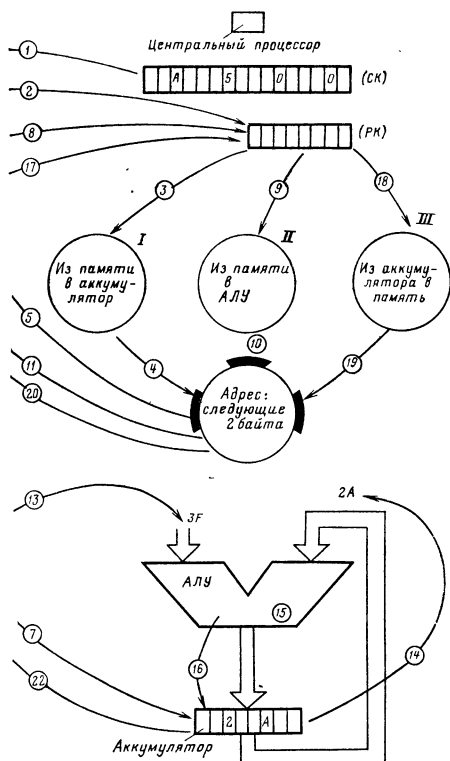
одно из двух чисел поместить в аккумулятор. Это выполняется следующей командой.

1. Послать содержимое ячейки памяти, адрес которой образован следующими за кодом операции (КОП) двумя байтами, в аккумулятор.

Этой команде на рис. 7.4 соответствуют шаги от 1 до 7. Код операции в этой команде обозначен как КОП-1. Сложение выполняется следующей командой.

II. Послать содержимое ячейки памяти, адрес которой задан в следующих двух байтах, в АЛУ и прибавить его к содержимому аккумулятора АЛУ. Поместить результат в аккумулятор.

Этой команде на рис. 7.4 соответствуют шаги 8—16. Код операции в этой команде обозначен как КОП-2. Результат



суммирования размещается в памяти с помощью следующей команды.

III. Послать содержимое аккумулятора в ячейку памяти, адрес которой определен следующими двумя байтами.

Этой команде на рис. 7.4 соответствуют шаги 17—22. Код операции обозначен как КОП-3. Программа размещается в ячейках памяти последовательно по возрастанию адресов, начиная с адреса А500.

Счетчик команд (СК) обеспечивает пошаговое выполнение программы. Счетчик команд похож на регулятора, который определяет последовательность выполнения событий. Перед началом выполнения программы в счетчик команд (рис. 7.4) следует занести значение А500. Теперь в нем содержится адрес первой команды.

1 — как только нажали кнопку ПУСК, программа начинает выполняться и происходит следующее.

2 — содержимое ячейки памяти по адресу А500 копируется и копия посылается в регистр команд.

3 — содержимое регистра команд становится равным КОП-1.

4 — КОП-1 декодируется и теперь ЭВМ знает, что следует выполнить команду 1.

5 — содержимое следующих 2 байт образует адрес ВС90.

6 — затем формируется адрес ячейки памяти ВС90.

7 — содержимое ячейки памяти ВС90 копируется и помещается в аккумулятор.

На этом выполнение команды I заканчивается.

Затем содержимое счетчика команд изменяется и становится равным 1503. Программа продолжает выполняться следующим образом.

8 — содержимое ячейки памяти 1503 копируется и помещается в регистр команд. Старое содержимое регистра команд (КОП-1) стирается.

9 — содержимое регистра команд равно КОП-2.

10 — КОП-2 декодируется и ЭВМ знает, что должна быть выполнена команда II.

11 — следующие 2 байта образуют адрес ВС91.

12 — идет обращение по адресу ВС91.

13 — содержимое ячейки памяти по адресу ВС91 передается в АЛУ.

14 — содержимое аккумулятора также пересылается в АЛУ.

15 — в АЛУ выполняется сложение.

16 — сумма (шестнадцатиричный код 69) посылается в аккумулятор. Старое содержимое аккумулятора (2А) стирается.

На этом заканчивается выполнение команды II.

Содержимое счетчика команд становится равным А506, и продолжается выполнение следующих операций.

17 — копируется содержимое ячейки памяти с адресом А506 и помещается в регистр команд. Прежнее содержимое регистра (КОП-2) стирается.

- 18 — содержимое регистра команд равно КОП-3.
19 — КОП-3 декодируется и теперь ЭВМ знает, что нужно выполнить команду III.
20 — следующие 2 байта образуют адрес ВС92.
21 — осуществляется адресация ячейки памяти ВС92.
22 — содержимое аккумулятора копируется и помещается по адресу ВС92.
На этом завершается выполнение команды III.

Выводы

1. Обрабатываемые данные хранятся в ОЗУ. Команды, из которых состоит программа, хранятся в ПЗУ.
2. Операции выполняются в АЛУ. Один из операндов должен быть помещен в аккумулятор. Результат операции остается в аккумуляторе.
3. Счетчик команд адресует ячейки памяти в последовательном возрастающем порядке.
4. Код операции помещается в регистр команд. Код операции декодируется. По результату декодирования ЭВМ знает, какое действие следует выполнить.

7.5. Ввод программ и данных

В предыдущих параграфах было установлено, что данные обрабатываются с помощью программы. Вопрос состоит в том, как ввести в основную память программу и данные.

Данные помещаются в ОЗУ. В большинстве случаев программа постоянно находится в ПЗУ. Обычно содержимое ПЗУ программируется изготовителем, но существуют ПЗУ, которые могут программироваться самим пользователем. Разумеется, программу можно было бы поместить в ОЗУ, но при этом возникает опасность ее стирания при отключении источника напряжения питания.

Если необходимо ввести данные в ячейку памяти, следует указать:

- а) адрес ячейки памяти, в которую посылается информация;
- б) число, которое должно быть помещено в ячейку памяти.

Послать информацию в память можно тремя различными способами:

- а) установив двоичный код с помощью переключателей;
- б) используя шестнадцатиричную клавиатуру;
- в) с помощью перфоленты.

В данной главе рассматриваются первые два способа. Ввод информации с перфоленты будет рассмотрен в гл. 18.

7.6. Ввод данных с помощью переключателей (рис. 7.5)

Индикаторные лампочки. Индикаторные лампочки не выполняют операцию ввода. Ряд из 16 лампочек представляет собой световой индикатор, показывающий, какая ячейка памяти адресуется. Индикатор из восьми лампочек показывает содержимое этой ячейки. В процессе работы программы содержимое индикаторов меняется очень быстро. В пошаговом режиме выполнения программы с помощью кнопки ШАГ световые индикаторы позволяют отслеживать содержимое ячеек памяти.

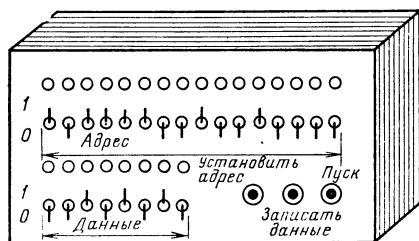


Рис. 7.5.

Переключатели. Для задания адреса ячейки памяти, в которую помещается информация, используется 16 двоич-

ных переключателей установки адреса. Если нажать кнопку **УСТАНОВИТЬ АДРЕС**, то происходит адресация соответствующей ячейки памяти. Информация, которую предполагается ввести в память, набирается с помощью восьми переключателей установки данных. Если нажать кнопку **ЗАПИСАТЬ ДАННЫЕ**, набранная информация попадает в ячейку с соответствующим адресом.

Таким образом, можно пословно ввести информацию в память.

Чтобы запустить программу, используются переключатели адресов памяти для установки адреса первого байта, с которого начинается программа. Затем нажимается кнопка **ПУСК** и программа начинает выполняться.

Ввод информации с помощью переключателей — очень трудоемкая работа. Кроме того, следует быть очень внимательным, так как легко перепутать 1 и 0.

7.7. Ввод данных с шестнадцатиричной клавиатуры

Процесс ввода занимает меньше времени и сопряжен с меньшим количеством ошибок, если для этой цели используется шестнадцатиричная клавиатура и если адреса и содержимое ячеек памяти считываются в шестнадцатиричной системе счисления. Соответствующий пример приведен на рис. 2.1. На рис. 7.6 показан фрагмент одноплатной микро-ЭВМ с клавиатурой.

Если нажать одну из клавиш 0—9 или А — F, то активируется соответствующая 4-битная комбинация. Ввод происходит следующим образом:

а) набираем адрес первой ячейки памяти;
б) нажимаем клавишу **УСТАНОВИТЬ АДРЕС**. В результате идет выборка набранного адреса;

в) набираем данные, которые должны попасть в ячейку памяти;

г) с помощью светового индикатора проверяем правильность задания адреса и данных (четыре световых индикатора слева указывают адрес; два световых индикатора справа указывают содержимое по этому адресу; два остальных индикатора показывают содержимое предшествующего адреса);

д) осуществляем ввод данных, нажимая клавишу **ЗАПИСЬ С ИНКРЕМЕНТИРОВАНИЕМ** (в результате в указанную ячейку посылается информация (запись) и адресуется следующая ячейка (увеличение адреса на единицу). Таким образом, достаточно указать только начальный адрес);

е) после того как осуществили ввод программы и данных, можно проверить правильность ввода, нажимая клавиши **ЧТЕНИЕ С ИНКРЕМЕНТИРОВАНИЕМ** и **ЧТЕНИЕ С ДЕКРЕМЕНТИРОВАНИЕМ** (при нажатии клавиши **ЧТЕНИЕ С ДЕКРЕМЕНТИРОВАНИЕМ** осуществляется выборка предыдущего адреса; при нажатии клавиши **ЧТЕНИЕ С ИНКРЕМЕНТИРОВАНИЕМ** выбирается следующий адрес);

ж) если окажется, что при вводе не совершили ни одной ошибки, то набираем адрес первого байта программы, нажимаем клавишу **УСТАНОВИТЬ АДРЕС** и затем клавишу **ПУСК**. Программа начинает выполняться;

з) после нажатия клавиши **ПУСК** клавиатура отключается от системы. Чтобы передать управление клавиатуре, следует нажать клавишу **СБРОС**, которая связана непо-

средственно с ЦП. В результате в счетчик адреса будет занесен код 0000. По этому адресу в ПЗУ расположена первая команда специальной программы (поставляемой из-

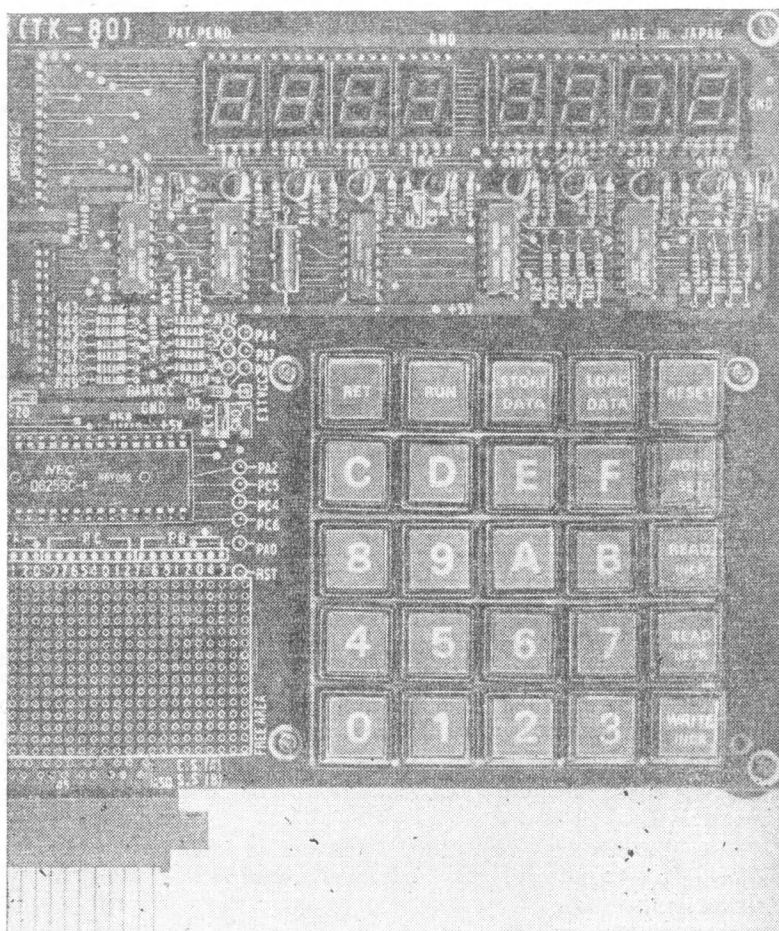


Рис. 7.6.

готовителем), которая осуществляет взаимодействие с клавиатурой. После нажатия клавиши СБРОС управление снова передается клавиатуре;

и) если хотим сохранить программу, ее можно записать на обычную магнитофонную кассету.

Клавиши **ЗАПОМНИТЬ ДАННЫЕ** и **ЗАГРУЗИТЬ ДАННЫЕ** служат соответственно для того, чтобы записать данные на кассету и переслать их в основную память. Направление обмена определяется по отношению к ЭВМ. Так, операция **ЗАПОМНИТЬ ДАННЫЕ** означает **ПЕРЕСЛАТЬ ДАННЫЕ ИЗ ОСНОВНОЙ ПАМЯТИ НА КАССЕТНОЕ ЗАПОМИНАЮЩЕЕ УСТРОЙСТВО**. Термин **ЗАГРУЗИТЬ ДАННЫЕ** означает **ПЕРЕСЛАТЬ ДАННЫЕ С КАССЕТНОГО ЗАПОМИНАЮЩЕГО УСТРОЙСТВА В ОСНОВНУЮ ПАМЯТЬ**.

Кассетный магнитофон подключен через схему, реализованную на печатной плате и размещенную рядом с клавиатурой.

П р и м е ч а н и е. В данной главе не будем рассматривать функции клавиши **ВОЗВРАТ**. Она имеет отношение к пошаговому режиму выполнения, который будет рассмотрен далее.

Выводы

1. Информацию в основную память можно поместить, используя переключатели, шестнадцатиричную клавиатуру, перфоленту.

2. Ввод информации с помощью переключателей не только связан с большими затратами времени, но и увеличивает вероятность ошибки.

3. Содержимое ячеек памяти может быть отображено с помощью лампочек (в двоичной системе) или 7-сегментных индикаторов (в шестнадцатиричной системе).

7.8. Работа клавиатуры и дисплея

Как видно из рис. 5.24, изготовитель помещает в ПЗУ микропроцессора специальную программу. Эта программа осуществляет управление вычислительной системой во время ввода рабочих программ. Эта управляющая программа называется монитором.

Монитор следит за тем, чтобы информация, поступающая на входы портов РА и РС, считывалась и передавалась в ЦП с периодом 1 мс. При нажатии некоторой клавиши двоичный код поступает на входные схемы и микро-ЭВМ узнает, какая клавиша была нажата. Если нажать кнопку 7, то

РА₇ подключается к РС₁₃ и единичный сигнал появляется на обоих входах.

Если была нажата одна из клавиш управления, например ЗАПИСАТЬ ДАННЫЕ или СЧИТАТЬ С ИНКРЕМЕНТИРОВАНИЕМ, монитор «знает», что выполняется соответствующая команда. Если нажата одна из клавиш 0 — F, то соответствующая битовая комбинация посылается в адресуемую ячейку ОЗУ и монитор преобразует эту битовую комбинацию в код 7-сегментного дисплея. Этот код затем посылается через буфер 8212 и драйверы в семисегментный дисплей. Переключение элементов 7-сегментного дисплея осуществляется последовательно, и поэтому не возникает ограничений на потребление тока. Более того, в каждый момент времени подается только один 8-битный код и поэтому достаточно иметь всего один драйвер. Интегральная микросхема IC2155, выполняющая роль дешифратора, определяет, какой дисплей должен отображать данную комбинацию 0 и 1. Семисегментные индикаторы дисплея работают последовательно, но с такой высокой частотой, что кажется, что они работают непрерывно. Микросхемы 8212 и 2155 должны управлять индикаторами синхронно, потому что в противном случае символ может появиться не на том знакоместе. Синхронизация осуществляется с помощью тактового генератора, реализованного на основе микросхемы 555.

Глава восьмая

АРХИТЕКТУРА ЦЕНТРАЛЬНОГО ПРОЦЕССОРА

Часть 1

8.1. Введение

Чтобы составить эффективные программы, необходимо познакомиться с архитектурой микропроцессора, т. е. с его функциональной организацией, которая представлена на рис. 8.1 и будет детально рассмотрена в этой главе.

Из рис. 8.1 видно, что микропроцессор 8-разрядной микро-ЭВМ в значительной степени состоит из регистров и шин, способных хранить и передавать слова длиной 8 бит. Регистры могут содержать результаты предыдущей операции, а также использоваться для временного хранения данных в течение периода исполнения какой-либо команды.

Прежде чем перейти к обсуждению функций, выполняемых различными регистрами, рассмотрим, какие действия совершаются в течение цикла выполнения команды (рис. 8.2).

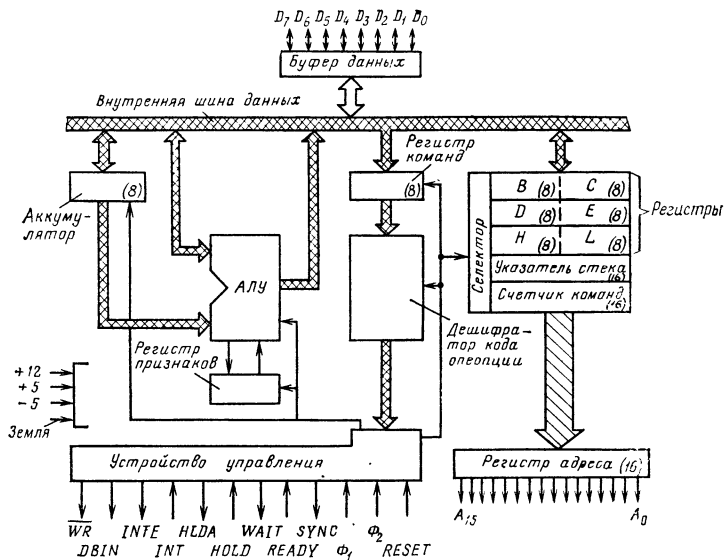


Рис. 8.1.

1. Команды могут состоять из 1, 2 или 3 байт, которые последовательно располагаются в памяти.

Во время выполнения предыдущей команды счетчик команд содержит значение, соответствующее адресу того

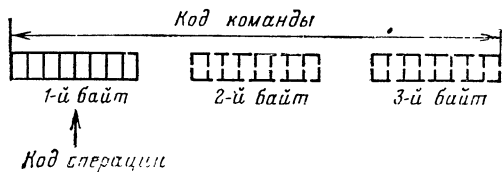


Рис. 8.2.

слова памяти, по которому располагается первый байт следующей предназначенной для исполнения команды.

2. Первый байт команды отводится для записи кода операции. Код операции указывает, какие действия должны

быть выполнены над данными, а также вид обрабатываемых данных.

3. Таким образом, цикл выполнения команды начинается со считывания из памяти первого байта команды, который содержит код операции.

4. Если в соответствии с кодом операции оказывается, что второй и третий байты команды вместе образуют адрес данных, предназначенных для обработки, то эти 2 байта должны быть переписаны в ЦП. После этого ЦП «знает», где следует искать требуемые данные.

5. Если же код операции непосредственно указывает место расположения данных, то их обработка может начинаться сразу после считывания первого байта. Код операции может, например, указывать, что обработке должен подвергнуться второй байт кода команды.

6. Итак, после считывания команды становится известно, где располагаются предназначенные для обработки данные и какая над ними должна выполняться операция. Так как первый байт команды содержит КОП, его передача из памяти в ЦП является частью общего процесса считывания команды. Если второй и третий байты содержат адрес данных, то считывание этих байтов из памяти в ЦП также является частью общего процесса считывания команды.

7. Выполнение команды сводится к выполнению соответствующих операций над данными. При выполнении команды также может производиться передача данных (например, из устройства ввода в ЦП).

8.2. Арифметическо-логическое устройство

В каждом микропроцессоре имеется *арифметическо-логическое устройство* (АЛУ), предназначенное для выполнения арифметических и логических операций.

8.2.1. Арифметические операции

Арифметическо-логические устройства различного типа способны выполнять разные наборы арифметических операций. Например, существуют микропроцессоры, АЛУ которых могут выполнять лишь операции сложения и сдвига. Очевидно, что, составив соответствующую программу, на основе этих основных арифметических операций можно выполнить и более сложные, такие как умножение и деление.

8.2.2. Логические операции

Арифметическо-логическое устройство практически любого микропроцессора способно выполнять логические операции И, ИЛИ и ИСКЛЮЧАЮЩЕЕ ИЛИ. Рассмотрим, каким образом их можно использовать.

Предположим, что на рис. 8.3, а представлено содержимое аккумулятора, и необходимо выяснить, содержимое разряда b_3 равно 0 или 1. Для этого с помощью операции И



Рис. 8.3.

необходимо сравнить биты аккумулятора с соответствующими битами числа, представленного на рис. 8.3, б. Все биты этого числа, называемого маской, равны 0, кроме бита, который должен сравниваться с b_3 и равен 1. Для битов от b_0 до b_2 и от b_4 до b_7 результатом сравнения будет 0. Если результат в целом есть 00001000, то b_3 имеет значение 1; если же результат есть 00000000, то b_3 равен 0. Таким образом, единицу из числа можно выделить с помощью маскирования всех других его бит. В гл. 11 эта логическая операция описана более детально.

8.2.3. Взаимосвязь регистров и АЛУ

Данные, предназначенные для обработки, или операнды, могут поступать в АЛУ одновременно из нескольких различных мест. Как видно из рис. 8.1, здесь возможны следующие случаи:

а) данные передаются в АЛУ из аккумулятора и, например, из регистра общего назначения D;

б) данные передаются из аккумулятора и через буфер шины данных из памяти.

Схема организации ЦП на рис. 8.1 соответствует так называемой одноадресной вычислительной машине. Другими словами, только один из операндов, передаваемых в АЛУ, поступает из аккумулятора. Другой операнд передается в АЛУ либо из одного из регистров общего назначения В, С, D, Е, Н или L, либо из памяти. Таким образом,

если необходимо сложить содержимое двух регистров общего назначения, то предварительно следует содержимое одного из них передать в аккумулятор.

Выводы

1. В АЛУ могут выполняться арифметические и логические операции.

2. Арифметическо-логические устройства большинства микропроцессоров способны выполнять следующие арифметические операции: сложение, вычитание и сдвиг.

С помощью этих базовых операций программист может написать программу, реализующую, например, умножение или деление.

3. Практически все АЛУ позволяют производить сравнение двух 8-разрядных чисел с помощью операций И, ИЛИ и ИСКЛЮЧАЮЩЕЕ ИЛИ.

8.3. Регистр признаков

Выполнение какой-либо операции может ставиться в зависимость от значения результата выполнения предыдущей операции. Подобная ситуация возникает, например, в том случае, когда при сложении появляется единица переноса. Чтобы можно было обратиться к информации о результатах

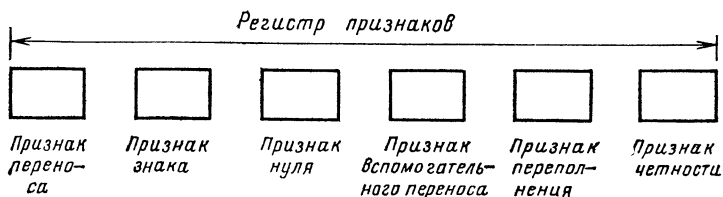


Рис. 8.4.

вычислений, АЛУ соединяется со специальным набором триггеров, которые устанавливаются в 1 или сбрасываются в 0 в зависимости от результата произведенных вычислений. Каждый из триггеров хранит какой-то один признак, а в совокупности эти триггеры образуют регистр признаков (рис. 8.4).

Примечание 1. Так как результаты всех операций, выполненных АЛУ, передаются в аккумулятор, можно

сказать, что регистр признаков содержит информацию о данных, пересылаемых из АЛУ в аккумулятор.

Примечание 2. Значения признаков используются только при выполнении команд определенного типа; в других случаях они игнорируются.

Примечание 3. В микропроцессорах разного типа используется разное число признаков.

8.4. Признак переноса

Одним из важнейших признаков является признак переноса. При сложении в АЛУ двух 8-разрядных чисел этот признак показывает, нужно ли переносить единицу в младший значащий разряд следующего байта.

Рассмотрим пример:

$$\begin{array}{r} 10111000 \\ 11011010 \\ \hline 10010010 \end{array} +$$

Признак переноса равен 1.

Признак переноса также указывает, нужно ли занимать единицу при вычитании двух 8-разрядных чисел. То, каким именно образом программист реализует вычитание, зависит от конкретного состава системы команд, который, в свою очередь, определяется конструкцией АЛУ. Если, например, АЛУ имеет аппаратные средства, позволяющие непосредственно реализовать операцию вычитания, в системе команд будет присутствовать команда SUB. Непосредственное выполнение операции вычитания иллюстрируется следующим примером:

$$\begin{array}{r} 11011111 \\ 10111001 \\ \hline 00100110 \end{array} -$$

Если АЛУ не содержит необходимых аппаратных средств, программист сначала должен позаботиться об образовании в АЛУ дополнительного кода числа, а затем выполнить сложение с помощью, например, команды ADD. Рассмотрим пример.

При вычитании из 11011111 числа 10111001 нужно сначала образовать обратный код вычитаемого (он равен 01000110), а затем его дополнительный код 01000111. Результат складывается с уменьшаемым:

$$\begin{array}{r} 11011111 \\ 01000111 \\ \hline 00100110 \end{array} +$$

В данном случае операция вычитания выполняется в два шага: сначала образуется дополнительный код числа, а затем выполняется сложение.

8.5. Признак вспомогательного переноса

Одним из вариантов признака переноса является признак вспомогательного переноса, который устанавливается в единицу, если в АЛУ происходит перенос из разряда b_3 в разряд b_4 . Этот признак используется при сложении чисел, записанных в коде BCD. Независимо от причины, порождающей перенос из разряда b_3 в b_4 , необходимо применение десятичной коррекции. Таким образом, при любой обработке чисел, записанных в коде BCD, программист должен учитывать в соответствующих командах возможность использования признака вспомогательного переноса. Если этого не делать, то признак автоматически игнорируется.

Далее приводится пример сложения чисел 19 и 09, записанных в коде BCD. Единица переноса из разряда b_3 в разряд b_4 в процессе сложения используется обычным образом. Вместе с тем эта единица воздействует на значение признака вспомогательного переноса, который устанавливается соответствующим образом. Это означает, что необходимо применение десятичной коррекции, которая выполняется автоматически в соответствии с содержащимися в команде указаниями (рис. 8.5).

Признак вспомо-	0001 1001	+	(BCD) = 19
могательного пе-	0000 1001	+	(BCD) = 09
реноса	<u>0010</u> <u>0010</u>		(BCD) \neq 28
	0010 0010	+	
	0000 0110	+	$\leftarrow +6$ — коррекция
	<u>0010</u> <u>1000</u>		(BCD) = 28

Рис. 8.5.

П р и м е ч а н и е. Применение десятичной коррекции также возможно при появлении одной из комбинаций от 1010 до 1111, которые, как известно, в коде BCD не используются.

8.6. Признак нуля

Признак нуля отмечает случай появления в АЛУ после выполнения какой-либо операции результата $00000000_{(2)}$. Этот признак используется, например, если нужно, чтобы вычислительная машина, прервав выполнение содержательной части программы, на некоторое время перешла в состояние ожидания. Для этого с помощью соответствующей команды помещаем в регистр, например, число 100. С помощью следующей команды из этого числа последовательно вычитается по единице до тех пор, пока результат не обратится в 0, в силу чего признак нуля будет установлен в 1. Только после этого будет выполняться следующая часть программы. Очевидно, что такой прием программирования позволяет строить циклы, в течение которых машина находится в ожидании.

Выводы

1. Триггер, который устанавливается в единицу или сбрасывается в нуль в зависимости от результата выполненной в АЛУ операции, хранит один из признаков. Совокупность таких триггеров образует регистр признаков.

2. Признак переноса указывает на факт появления единицы переноса или займа единицы при сложении или вычитании двух чисел.

3. Признак вспомогательного переноса указывает, что во время сложения произошел перенос из разряда b_3 в разряд b_4 . Этот признак используется при выполнении вычислений над числами, записанными в коде BCD; он показывает, что необходимо применение десятичной коррекции.

4. Признак нуля показывает, что результат вычислений в АЛУ есть $00000000_{(2)}$.

8.7. Признак знака

Как было показано в гл. 4, отрицательные числа представляются в вычислительной машине в виде дополнительных кодов. В этом случае старший значащий разряд может нести не только какое-то цифровое значение, но и знак числа: если в старшем значащем разряде стоит 1, то число отрицательное, если — 0, то число положительное. В силу того, что значение старшего значащего разряда определяет знак числа, старший значащий разряд результата выполненной в АЛУ операции запоминается в признаке знака для дальнейшего использования.

8.8. Признак переполнения

Вычисления с использованием дополнительных кодов производятся над словами определенной длины. Если в процессе вычислений получается результат большей длины, то должен вырабатываться сигнал, требующий расширения длины слова. Если такое расширение невозможно, то вычисления должны останавливаться. Средством индикации того, что возникла подобная ситуация, является признак переполнения.

Не будем обсуждать здесь, как именно в АЛУ решается вопрос добавления к слову дополнительного байта.

При работе с микро-ЭВМ, в которой не предусмотрено наличие признака переполнения, длину слова необходимо выбирать так, чтобы ее было достаточно как для представления операндов, так и для представления результата операции. Если же в этом нет уверенности, программа должна содержать средства автоматического контроля длины слова и ее адаптации к возможному переполнению.

8.9. Признак четности

Этот признак устанавливается равным единице, если в результате операции общее число единиц является четным. Например, если результат вычислений представляет собой значение 01101010, то происходит установка признака четности, так как здесь имеются четыре единицы — четное число. Признак четности, как правило, используется для контроля на четность данных при их передаче; он позволяет выявить ошибки, которые при этом, возможно, возникнут.

8.10. Регистр команды и дешифратор кода операции

Вид каждой операции, выполняемой микро-ЭВМ, определяется с помощью кода соответствующей команды.

Если в микро-ЭВМ отводится 8 бит для записи кода операции, то оказывается возможным различать максимум $2^8 = 256$ кодов операций. В большинстве случаев такого числа операций более чем достаточно. Число байт, отводимых для записи команды, определяется типом операции (рис. 8.6). В микро-ЭВМ максимальное число этих байт обычно равно трем.

Код операции, указывающий, как происходит обработка данных при выполнении команды, всегда размещается в первом байте. Если вся команда занимает 1 байт, то для кода операции отводится часть этого байта, например 2 бита. Если команда занимает 3 байта, то для кода операции отводится весь первый байт.

Весь первый байт кода команды считывается из памяти и передается в регистр команды в течение цикла считывания независимо от того, какая его часть отведена для записи кода операции. Декодирование содержимого первого байта позволяет определить следующее:

а) сколько байтов содержится в команде;

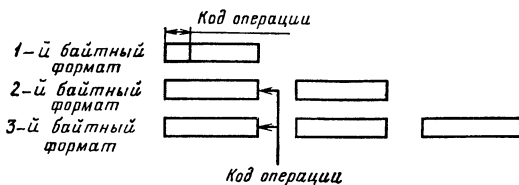


Рис. 8.6.

б) является ли содержимое второго и третьего байтов в совокупности адресом памяти, по которому хранятся предназначенные для обработки данные;

в) какая должна выполняться операция.

Для декодирования первый байт передается из регистра команды в дешифратор кода операции, по результатам работы которого под воздействием тактовых сигналов вырабатывается нужная последовательность сигналов управления. Это приводит к считыванию второго и третьего байтов из памяти, если это необходимо, а также к собственно выполнению операции, предписываемой командой.

8.11. Регистры общего назначения

На структурной схеме центрального процессора (см. рис. 8.1) изображен определенный набор регистров общего назначения. Эти регистры обеспечивают быстрый доступ к хранящимся в них операндам.

Для адресации регистров общего назначения и аккумулятора используется укороченное адресное поле длиной 3 бита. С помощью слова такой длины можно различать максимум $2^3 = 8$ регистров. На рис. 8.1 регистры общего

назначения обозначены буквами В, С, D, Е, Н и L. Из рис. 8.1 видно, что эти регистры объединены попарно, что позволяет обрабатывать слова длиной как 8 бит, так и 16 бит. Последняя возможность используется, если необходимо организовать хранение адреса памяти в регистрах или выполнить вычисления над 2-байтными числами.

Выводы

1. В признаке знака копируется значение старшего значащего разряда результата вычислений над дополнительными кодами. Признак знака показывает, каким является число: положительным или отрицательным.

2. Признак переполнения является средством, указывающим на возникновение необходимости увеличить длину слова и привести ее в соответствии с длиной результата вычислений.

3. Признак четности показывает, какое число единиц содержит результат какой-либо операции: четное или нечетное.

4. Тот байт кода команды, который содержит код операции, всегда записывается в регистр команды.

5. Число байт, составляющих код команды, определяется типом выполняемой операции. У 8-разрядных микроЭВМ код команды имеет длину 1, 2 или 3 байта.

6. В ЦП регистры общего назначения обеспечивают более быстрый доступ к операндам, так как при обращении к операндам этап считывания из памяти оказывается уже выполненным.

8.12. Счетчик команд

Счетчик команд указывает, где в памяти расположены байты данной команды. Устройство управления увеличивает содержимое счетчика команд на единицу всякий раз, когда байт кода команды передается из памяти в ЦП. Если код команды состоит из 2 байт, то ее считывание происходит за два шага. Перед началом считывания счетчик команд уже содержит адрес байта текущей команды, так как содержимое его было увеличено на единицу; в конце процедуры считывания предыдущей команды первый байт сразу может передаваться в ЦП, после чего содержимое счетчика команд снова увеличивается на единицу. Теперь счетчик содержит адрес второго байта текущей команды,

после передачи которого в ЦП содержимое счетчика команд опять увеличивается на единицу и определяет адрес первого байта следующей команды.

8.13. Указатель стека

Чтобы объяснить значение термина «указатель стека», сначала необходимо познакомиться, что представляют собой команды передачи управления (см. гл. 2) и подпрограммы.

8.13.1. Команды передачи управления

Команды, образующие любую программу, располагаются в памяти последовательно в порядке возрастания ее адресов. Обычно команды выполняются в той же последовательности, в которой они расположены в памяти. Нарушить этот порядок программист может с помощью команды передачи управления.

При выполнении процедуры считывания из памяти команды передачи управления содержимое счетчика команд трижды увеличивается на единицу. Когда эта команда выполняется, содержимое счетчика команд заменяется тем адресом, который содержится в команде передачи управления. На рис. 8.7 условно изображена программа, содержащая две такие команды.

Выполнение или невыполнение команды передачи управления иногда ставится в зависимость от значения одного из признаков. Например, команда передачи управления может выполняться, если результат предыдущей операции равен нулю; если результат не равен нулю, то выполняется следующая по порядку команда программы.

Если передача управления ставится в зависимость от состояния признака, то тогда говорят об условном переходе или ветвлении. Если же передача управления выполняется всегда, при любых условиях, то говорят о безусловном переходе.

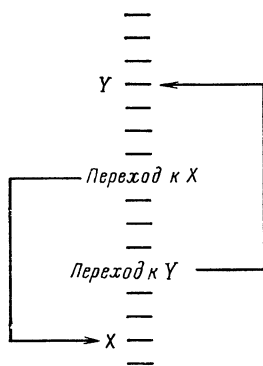


Рис. 8.7.

8.13.2. Подпрограммы

В том случае, когда происходит обращение к подпрограмме, применяют специальные команды передачи управления.

Подпрограммой называют такую часть основной программы, которая в ходе выполнения этой основной программы повторяется несколько раз. Например, в программе может возникнуть необходимость многократного перемножения двух чисел, причем, хотя сами числа могут изменяться, процедура их перемножения остается одинаковой во

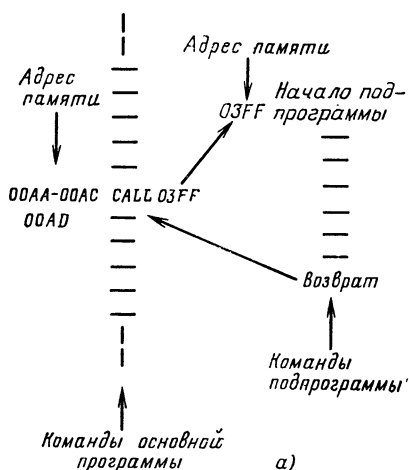
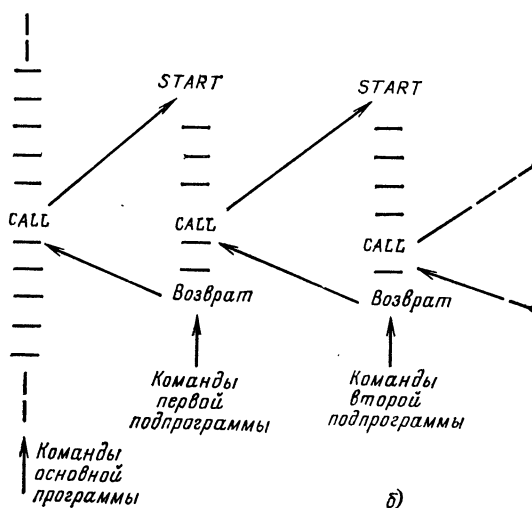


Рис. 8.8.



всех случаях. Поэтому писать программу перемножения и хранить в памяти соответствующую последовательность команд для каждой возможной пары чисел крайне неэф-

фективно. Неизменность совершаемой над данными процедуры позволяет соответствующую последовательность команд помещать в память в единственном экземпляре. Обращение к этой последовательности команд, которая и называется подпрограммой, выполняется по мере необходимости (рис. 8.8, а). Если необходимо в основной программе перемножить два числа, можно обратиться к подпрограмме с помощью команды CALL. Эта команда содержит адрес первой выполняемой команды подпрограммы [в данном случае $03FF_{(16)}$] и занимает 3 байта.

В процессоре имеются специальные средства, с помощью которых организуется возврат к основной программе после завершения выполнения подпрограммы. Всякий раз, когда процессор получает для исполнения команду CALL, содержимому счетчика команд дается приращение, после чего это содержимое записывается в специальной области памяти [в рассматриваемом примере по адресу $00AD_{(16)}$], зарезервированной для этой цели программистом и называемой стеком. Таким образом, стек содержит адрес команды, предназначенной для исполнения ЦП после завершения подпрограммы. Заметим, что в некоторых микропроцессорах роль стека играет не часть основной памяти, а отдельное ОЗУ, которое определенным образом соединяется с микропроцессором.

После этого ЦП замещает содержимое счетчика команд адресом первой команды подпрограммы, который указывается в команде CALL [это адрес $03FF_{(16)}$]. Это означает, что следующей командой, которую процессор считает из памяти, будет первая команда подпрограммы.

Последняя команда подпрограммы — это всегда команда возврата, в которой никакой адрес не содержится. Команда возврата обеспечивает перезапись адреса, хранящегося в стеке [в данном примере $AD_{(16)}$], обратно в счетчик команд. После этого выполнение основной программы может продолжаться с того места, в котором произошло обращение к подпрограмме.

Часто оказывается, что подпрограммы вложены одна в другую (рис. 8.8, б). Это означает, что в одной подпрограмме может содержаться обращение ко второй, которая, в свою очередь, обращается к третьей и т. д. Если стек способен хранить, например, три адреса возврата, то можно осуществить два вложения, т. е. число возможных вложений всегда на единицу меньше числа адресов возврата, которые могут содержаться в стеке.

8.13.3. Указатель стека

В ЦП имеется специальный 16-разрядный регистр, который предназначен для адресации ячеек стековой памяти и носит название указателя стека. Указатель стека определяет адрес тех ячеек памяти в стеке, в которых хранится нужный адрес возврата.

8.14. Регистр адреса

Чтение и запись информации в память может происходить, если определено значение соответствующего адреса памяти. Этот адрес указывает ячейку памяти, предназначенную для записи или считывания байта команды или байта данных. Центральный процессор передает адрес из регистра в память по шине адреса. Для доступа к памяти требуется некоторое время, в силу чего возможность обратиться к нужному слову в памяти появляется не сразу. Существование такой задержки обуславливает необходимость хранения адреса, сформированного центральным процессором, в течение определенного промежутка времени. Чтобы это было возможно, в большинстве микро-ЭВМ встраивается специальный регистр, предназначенный для хранения адреса памяти и называемый регистром адреса.

Выводы

1. В команде передачи управления содержится адрес следующей исполняемой команды, которая, следовательно, может храниться в произвольном месте памяти.

2. Подпрограммой называется последовательность команд, которая может выполняться многократно по ходу основной программы.

3. Для обращения к подпрограмме используется команда CALL. В ней содержится адрес первой исполняемой команды подпрограммы. Последней командой подпрограммы является команда возврата.

4. Стекком называют часть памяти, в которой, например, могут храниться адреса возврата из подпрограмм.

5. Указатель стека — это 16-разрядный регистр в ЦП, адресующий ячейки памяти в стеке.

6. Регистр адреса ЦП предназначен для хранения адресов памяти в течение того времени, которое требуется для их декодирования.

Глава девятая

АРХИТЕКТУРА ЦЕНТРАЛЬНОГО ПРОЦЕССОРА

Часть 2

9.1. Генератор тактовых сигналов

Необходимость того, чтобы в микро-ЭВМ события происходили в нужной последовательности и скоординированно, требует организации управления событиями во времени, что характерно и для любого другого цифрового устройства. Такое управление требует наличия в составе микро-ЭВМ интегрального генератора тактовых сигналов. Тактовый генератор часто размещают на том же кристалле, что и микропроцессор. Однако в микро-ЭВМ, изображенной на рис. 2.1, тактовый генератор не входит в состав БИС микропроцессора, а выполнен в виде самостоятельной ИС типа 8224 (см. также рис. 5.24).

Сигналы тактирования, обычно обозначаемые Φ , могут

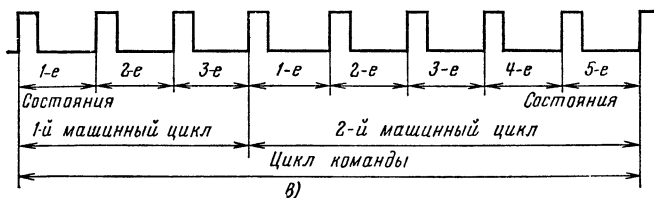
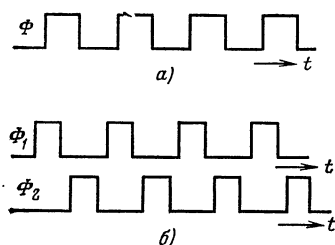


Рис. 9.1.

представлять собой последовательность прямоугольных сигналов, подобную той, которая изображена на рис. 9.1, а. В большинстве случаев, однако, используется пара сигналов тактирования: Φ_1 и Φ_2 , представляющих собой две последовательности прямоугольных сигналов с одинаковой амплитудой, частотой и скважностью, но сдвинутых по фазе на 180° (рис. 9.1, б). Прежде, чем подробнее познакомимся с назначением сигналов тактирования, объясним смысл понятий: цикл команды, машинный цикл и состояние.

Циклом команды называют время, необходимое для считывания команды из памяти и ее исполнения. Цикл команды реализуется за 1—5 машинных циклов, точное число которых зависит от сложности команды и равно числу обращений ЦП к памяти или одному из устройств ввода-вывода. Таким образом, можно констатировать, что, во-первых, число машинных циклов в цикле команды определяется тем, сколько раз используется шина данных, и, во-вторых, цикл любой команды состоит, по меньшей мере, из одного машинного цикла, так как даже в самом простом случае необходимо извлечь из памяти 1 байт команды и передать его в ЦП.

В свою очередь, каждый машинный цикл состоит из определенной последовательности элементарных действий, называемых состояниями (тактами). Например, для считывания команды необходимо сначала определить значение нужного адреса памяти и декодировать его. Только после этого первый байт команды можно передавать в ЦП и записывать в регистр команды. Таким образом, состояние — это простейшее действие, которое может быть выполнено в микро-ЭВМ. Состояние выполняется в течение одного периода сигнала тактирования, а в отдельном машинном цикле может быть от трех до пяти состояний.

Для определения времени выполнения команды нужно знать, какое число состояний содержится в цикле команды и чему равен период сигнала тактирования. Например, команда ADD (см. гл. 2) содержит четыре состояния. Если при этом период сигнала тактирования равен 500 нс, то цикл команды потребует 2 мкс; таким методом можно определить как время цикла любой команды, так и время, необходимое для выполнения всей программы. На рис. 9.1, в приведен пример временной структуры цикла команды и составляющих его машинных циклов в «привязке» к сигналу тактирования.

9.2. Устройство управления

Устройство управления является одним из важнейших блоков ЦП. Совместно с генератором тактовых сигналов устройство управления обеспечивает, чтобы события в микро-ЭВМ происходили в правильной последовательности.

После извлечения команды из памяти и ее дешифрирования устройство управления генерирует последовательность сигналов, необходимую для выполнения команды.

Во многих микро-ЭВМ устройство управления, кроме того, способно самостоятельно реагировать на различные внешние сигналы. Например, при поступлении сигнала прерывания от телетайпа в связи с необходимостью ввода данных устройство управления прерывает выполнение основной программы и вводит в действие нужную стандартную подпрограмму.

Сигнал готовности, поступающий из памяти или порта УВВ, также воспринимается устройством управления. Использование сигналов готовности необходимо в тех случаях, когда память или УВВ проигрывают в быстродействии центральному процессору. В этом случае может оказаться, что ЦП должен ждать, пока данные в памяти или порте УВВ не станут доступными для передачи.

Выводы

1. *Циклом команды* называют время, необходимое при выполнении команды для считывания из памяти и ее исполнения.

2. Цикл команды состоит из 1—5 машинных циклов. Машинный цикл состоит из 3—5 состояний.

3. Число машинных циклов равно числу обращений ЦП к памяти или одному из устройств ввода-вывода.

4. *Состояние* — это простейшее действие, которое может быть выполнено в микро-ЭВМ. Состояние выполняется в течение одного периода сигналов тактирования.

5. Устройство управления обеспечивает смену состояний в правильной последовательности и в «привязке» к сигналам тактового генератора.

9.3. Временные диаграммы

Характеристики МП обычно публикуются их изготовителями в виде временных диаграмм, на которых последовательность событий представляется как функция времени, причем временную диаграмму можно построить для любой операции, выполняемой микро-ЭВМ.

В качестве примера рассмотрим временную диаграмму выполнения команды ввода (рис. 9.2, б).

Чтобы можно было понять эту диаграмму, сначала необходимо точно определить формат команды ввода (рис. 9.2, а).

Команда занимает два байта, первый из которых, содержащий код операции, указывает, какие действия нужно

совершить (принять данные из порта ввода). Второй байт указывает на операнд, т. е. из какого порта должны быть приняты данные.

На рис. 9.2, б изображены:

- а) сигнал тактирования Φ_1 ;
- б) сигнал тактирования Φ_2 , изменяющийся в противофазе с Φ_1 ;
- в) сигнал синхронизации SYNC, порождаемый сигналом Φ_2 .

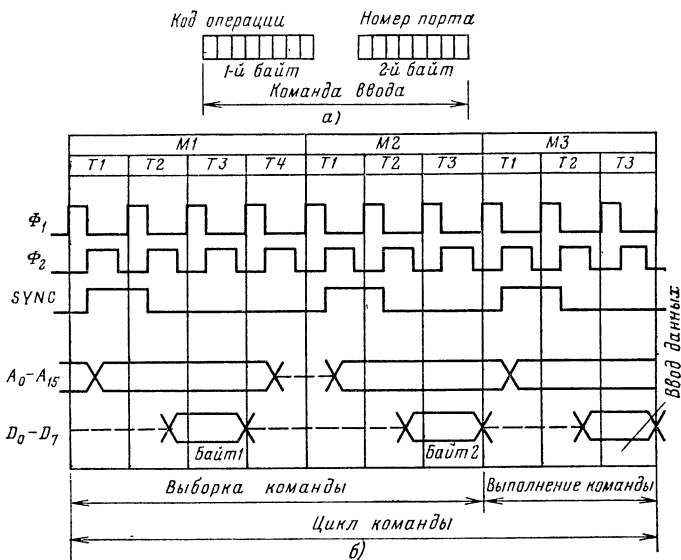


Рис. 9.2.

Появление каждого сигнала SYNC означает начало нового машинного цикла;

г) наличие или отсутствие передачи сигналов $A_0—A_{15}$ по шине адреса. Так как для общего случая невозможно указать конкретные значения этих сигналов, ограничимся только указанием того, что по шине адреса передается или не передается некоторая информация. Как видно из рис. 9.3, отсутствие информации, передаваемой по шине адреса, обозначается пунктирной линией;

д) наличие или отсутствие передачи сигналов $D_0—D_7$ по шине данных.

На временной диаграмме рис. 9.2, б изображены три машинных цикла: М1—М3. Код операции считывается в течение цикла М1. В течение цикла М2 адрес операнда считывается из памяти, а в течение цикла М3 происходит выполнение команды, т. е. по адресу порта входятся данные, считываемые и передаются в ЦП. Таким образом, считывание команды производится в течение циклов М1 и М2, а ее исполнение в течение цикла М3.

Каждый машинный цикл состоит из определенного числа состояний. Код адреса передается по шине адреса в течение состояния Т2 каждого машинного цикла. В цикле М1 — это адрес кода операции в памяти, в цикле М2 — это адрес операнда (номер порта) в памяти, в цикле М3 — это номер нужного порта ввода.

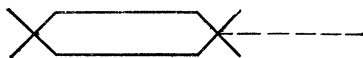


Рис. 9.3.

Во время состояния Т2 каждого машинного цикла производится проверка условий, которые могут сделать необходимой задержку в выполнении данного машинного цикла. Одной из причин, которые могут вызвать задержку, может быть разница в быстродействии ЦП и порта УВВ или ЦП и памяти.

Если существуют причины для такой задержки, то ЦП переходит в состояние ожидания; если же причин для задержки нет, то первый и второй байты команды считываются из памяти в течение состояний Т3 машинных циклов соответственно М1 и М2. Эта информация передается в ЦП по шине данных в виде сигналов D_0 — D_7 . В течение состояния Т3 машинного цикла М3 команда выполняется, т. е. данные принимаются из порта ввода.

Из временной диаграммы также видно, что машинный цикл М1 содержит не три, а четыре состояния. Четвертое состояние отводится для таких действий ЦП, как дешифрирование кода операции.

Ввод информации связан с обращением к порту УВВ, в результате чего для выполнения команды ввода требуется добавить отдельный машинный цикл. В том случае, когда выполнение команды реализуется исключительно средствами ЦП, а необходимость обращения к памяти или УВВ отсутствует, выполнение команды может происходить в течение четвертого или, возможно, пятого состояния предшествующего машинного цикла. Следовательно, существует возможность реализации цикла команды за один машинный цикл.

Примером этого может служить команда ADD r, рассмотренная в одной из предыдущих глав. Команда состоит из одного байта и полностью выполняется в ЦП, так как при выполнении этой команды производится сложение содержимого регистра общего назначения r с содержимым аккумулятора. Это и позволяет реализовать цикл команды за один машинный цикл (необходимость обращения к памяти возникает только на этапе считывания команды).

9.4. Внешние выводы БИС микропроцессора

Рассмотрим типовую конфигурацию системы внешних вызовов МП (рис. 9.4), отмечая одновременно те особенности, которые могут встретиться в некоторых случаях.

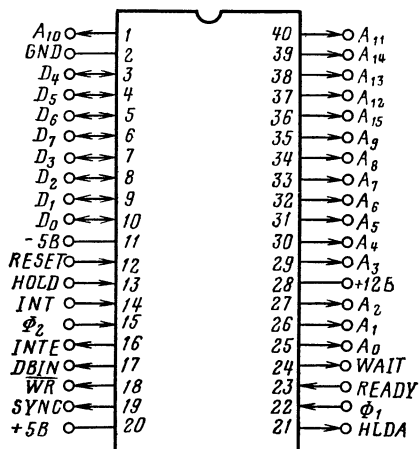


Рис. 9.4.

Все МП должны иметь соединение с шиной адреса, которая используется для передачи адресов памяти или одному из УВВ. Так как большинство МП работает с 16-разрядными адресами, необходимо предоставить в распоряжение ЦП 16 внешних выводов A₀ — A₁₅. Существуют МП, в которых некоторые из этих выводов выполняют двойную функцию: по ним передаются либо значения соответствующих бит

адреса, либо информация о процессах записи-считывания в памяти. Иногда восемь из шестнадцати выводов шины адреса могут использоваться для подключения к шине данных в режиме мультиплексирования. D₀ — D₇ — выходы двунаправленной шины данных. Каждый 8-разрядный МП имеет восемь выводов для подключения шины данных. Так как шина данных является двунаправленной, эти выходы используются для передачи информации и в ЦП, и из него; Φ₁, Φ₂ — тактовые входы. На выводы 22 и 15 (рис. 9.4) подаются сигналы тактирова-

ния Φ_1 и Φ_2 . Заметим, что в рассматриваемом МП сигналы тактирования генерирует внешняя ИС. На некоторые МП требуется подавать лишь сигнал тактирования Φ_1 , в то время как сигнал Φ_2 вырабатывается с помощью сигнала Φ_1 самим микропроцессором.

SYNC — *выход синхронизации*. С вывода 19 (рис. 9.4) снимается сигнал синхронизации. Этот сигнал оповещает память и УВВ о начале нового машинного цикла, чем обеспечивается согласование работы этих устройств с работой ЦП во времени.

Выводы для подключения напряжения питания. На любой МП должно быть подано напряжение питания. Через соответствующие выводы ЦП, изображенный на рис. 9.4, подключается к источникам питания +12, -5, +5 В и общей точке GND (ground). Другие типы МП подключаются только к источнику питания +5 В и общей точке GND.

RESET — *вход гашения*. По этому входу можно сбрасывать счетчик команд, т. е. устанавливать его значение равным 0000₍₁₆₎, и затем передавать в регистр адреса нулевой адрес первой исполняемой команды. Заметим, что после подачи сигнала гашения RESET содержимое всех других регистров ЦП остается неопределенным.

WR — *выход сигнала «Запись»*. Информация передается по шине в двух направлениях: из ЦП в память или порт УВВ либо в противоположном направлении. Поэтому ЦП должен информировать память или порт УВВ о выполнении им операций записи (WRITE) или чтения (READ).

Центральный процессор указывает на то, что выполняется операция записи путем подачи сигнала высокого или низкого уровня на вывод WR (write). Некоторые МП при записи в память или выводе информации генерируют сигнал WRP (write pulse), что позволяет синхронизировать работу памяти или УВВ. **READY** — *вход сигнала готовности*. Если необходимо произвести обмен данными между памятью или УВВ и ЦП, то перед началом передачи данных следует указать адрес модуля, которому посылается информация или, наоборот, от которого она принимается. Этот адрес декодируется в памяти или в порте УВВ, что позволяет определить источник либо место назначения передаваемых данных. Однако, прежде чем сможет действительно начаться обмен данными, проходит некоторый промежуток времени, который называют временем доступа. То, что этот промежуток времени истек, отражается в посылке памятью

или портом УВВ сигнала готовности на соответствующий вход микропроцессора. Это дает ЦП информацию о том, что может начаться обмен данными.

WAIT — выход сигнала ожидания. Центральный процессор указывает, что он находится в состоянии ожидания (например, в течение времени доступа), путем подачи сигнала на вывод WAIT.

DBIN — выход сигнала «шина данных в режиме ввода». Если в ЦП передаются данные при считывании их из памяти или при выполнении операции ввода, то он указывает на это путем генерации сигнала на выводе DBIN (data bus in).

Таким образом, при считывании из памяти сигнал DBIN отображает команду READ, а при выполнении операции ввода команду IN.

HOLD — вход сигнала захвата шин и HLDA — выход сигнала подтверждения захвата шин. Большие объемы информации, хранящиеся, например, на гибких магнитных дисках, могут передаваться в основную память микро-ЭВМ без вмешательства ЦП. Такой режим работы называют прямым доступом к памяти (ПДП). В режиме ПДП, однако, ЦП должен отключаться от шин адреса и данных. Другими словами, выходы $D_0 - D_7$ и $A_0 - A_{15}$ должны переходить в высокоимпедансное состояние. Это происходит при появлении сигнала захвата шин на входе HOLD. Результатом появления сигнала захвата является то, что ЦП отключается от своих шин и информирует об этом другие блоки с помощью сигнала подтверждения захвата шин, подаваемого на выход HLDA (hold acknowledge output).

В некоторых МП выходы HOLD и HLDA носят соответственно названия «вход сигнала, пауза» и «выход, работа/пауза».

INT — вход сигнала «запрос прерывания» и INTE — выход сигнала разрешения прерывания. Если, например, возникает необходимость передачи данных от УВВ, то соответствующее устройство посылает запрос прерывания центральному процессору. Этот сигнал поступает на вывод INT (interrupt).

В микропроцессоре 8080 (рис. 9.4) предусмотрена возможность использования специальной команды, запрещающей ЦП реагировать на запрос прерывания. Если такой запрет отсутствует, то МП выдает сигнал разрешения прерывания на вывод INTE (interrupt enable).

Другими словами, если сигнал INTE равен единице, то ЦП принимает запрос прерывания; если же сигнал INTE

равен нулю, то запрос прерывания от УВВ игнорируется центральным процессором.

В некоторых МП сигналу INTE соответствует сигнал «Подтверждение прерывания», появляющийся на выводе INTA (interrupt acknowledge).

Выводы

1. Характеристики МП отражаются на временной диаграмме его работы. На временной диаграмме последовательность событий представляется как функция времени.

2. Восьмиразрядный МП имеет восемь выводов для подключения шины данных. Большинство 8-разрядных МП имеет 16 выводов для подключения шины адреса. В ряде МП некоторые выводы могут выполнять двойную функцию.

3. Кроме выводов, предназначенных для подключения шин данных и адреса, любой МП имеет набор входов и выходов, через которые передаются сигналы управления.

Глава десятая

АРХИТЕКТУРА МИКРО-ЭВМ

10.1. Введение

Микро-ЭВМ состоит из четырех блоков: центрального процессора (ЦП); основной памяти, которая, в свою очередь, состоит из одной или нескольких БИС памяти; портов УВВ; системы межмодульных связей.

В настоящей главе рассматриваются структурные схемы этих устройств, причем схемы ЦП, который был подробно проанализирован в предыдущих главах, коснемся весьма поверхностно. В конце данной главы приводится структурная схема микро-ЭВМ, с помощью которой можно проследить потоки данных, возникающие в микро-ЭВМ при выполнении команд.

10.2. Структурная схема центрального процессора

Структурная схема ЦП, рассмотренная в гл. 9, представлена на рис. 10.1 с добавлением следующих элементов:

1) регистра временного хранения информации ТЕМ, предназначенного для краткосрочного хранения данных перед их передачей в АЛУ или один из регистров ЦП;

2) регистров общего назначения W, Z, предназначенных для краткосрочного хранения данных во время выполнения команды.

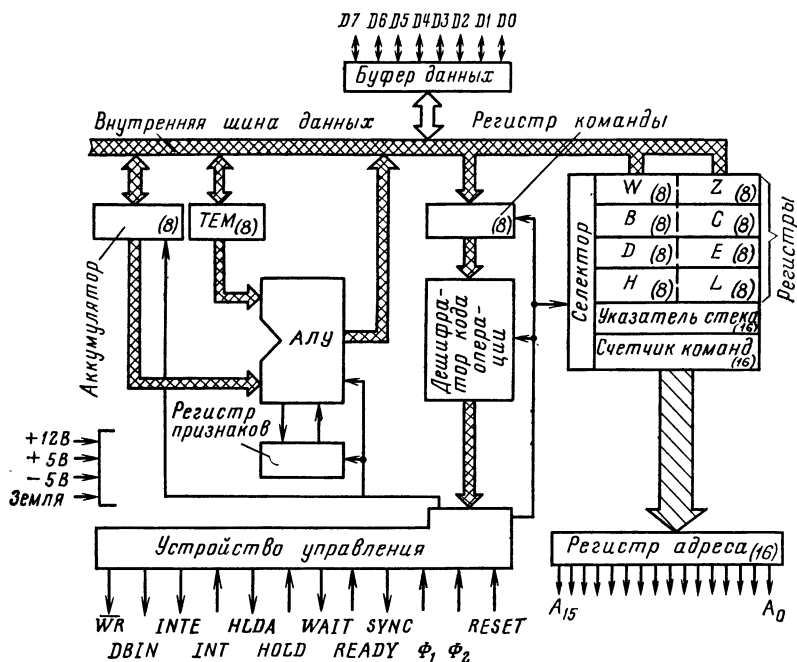


Рис. 10.1.

Регистры временного хранения информации относятся к программно-недоступным узлам ЦП, поэтому они не рассматривались в гл. 8. Однако об этих регистрах невозможно не упомянуть, говоря о потоках данных, возникающих при выполнении команд.

10.3. Структурная схема памяти

Структурная схема памяти изображена на рис. 10.2. Любое устройство памяти состоит из большого числа элементов, каждый из которых может хранить 1 бит информации. В 8-разрядной микро-ЭВМ восемь элементов образуют одну ячейку памяти, в которой может храниться 1 байт данных. Число адресуемых и, следовательно, доступных

для программиста ячеек памяти определяется разрядностью адреса памяти. Очевидно, что если ЦП, как это показано на рис. 10.2, работает с 16-разрядными адресами памяти, то нет никакого смысла при этом использовать память объемом более $2^{16} = 65\,536$ слов.

Адресный коммутатор (рис. 10.2) предназначен для селекции ячейки памяти с заданным адресом с целью передачи содержимого ячейки в шину данных, причем каждый из восьми элементов ячейки памяти подключается к соответствующей линии шины данных.

Кроме адреса, ЦП посылает в память сигнал управления по предназначенной для этого линии управления. Сигнал управления определяет характер действий (считывание или запись), выполняемых при данном обращении к памяти. По окончании селекции слова с заданным адресом память выдает сигнал в линию READY (ГОТОВ) шины управления. При записи информации в память этот сигнал указывает на то, что данные могут вводиться в память; если выполняется операция считывания из памяти, то сигнал готовности разрешает ЦП начать прием данных.

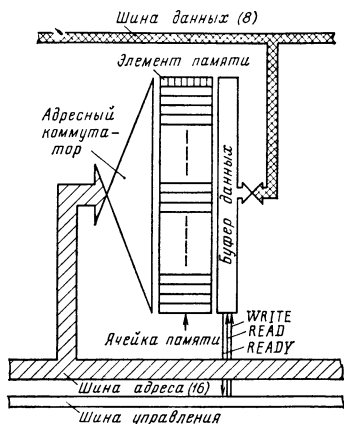


Рис. 10.2.

Выводы

1. Микро-ЭВМ состоит из центрального процессора, основной памяти, портов УВВ, системы межмодульных связей.

2. Память состоит из большого числа элементов, каждый из которых хранит 1 бит информации. В 8-разрядной микро-ЭВМ восемь элементов образуют одну ячейку памяти.

3. Центральный процессор посылает в память наряду с адресом сигнал записи или считывания, с помощью которого определяется направление передачи данных: из ЦП в память или из памяти в ЦП.

4. Память указывает на окончание процесса селекции слова по заданному адресу с помощью сигнала готовности.

10.4. Структурная схема модуля ввода-вывода

Возможность обмена данными между ЦП и внешней средой обеспечивается наличием в составе микро-ЭВМ модулей ввода-вывода. Модуль ввода-вывода (рис. 10.3) состоит из следующих компонентов:

1) схемы селекции УВВ, предназначенной для декодирования адреса (номера порта), заданного ЦП, и выбора нужного порта для ввода или вывода информации;

2) набора портов вывода, представляющих собой регистры, в которых принятые из ЦП данные хранятся до тех пор, пока они не смогут быть переданы устройству вывода;

3) набора портов ввода (буферов), через которые данные передаются в ЦП. При выполнении операции ввода сигналы пропускаются через выбранный порт только в течение короткого промежутка времени.

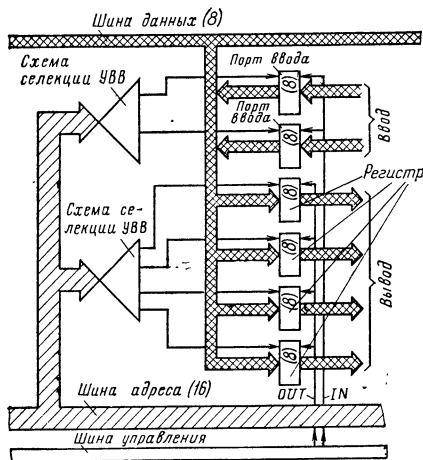


Рис. 10.3.

В процессе вывода информации схемы ввода-вывода выполняют следующие операции.

1. По заданному центральным процессором адресу (номеру) порта производится селекция того порта вывода, который должен принять данные, передаваемые из ЦП.

2. По команде OUT данные передаются из ЦП в порт вывода.

3. Порт вывода должен принять данные, после чего они становятся доступными для внешней среды.

В процессе ввода информации выполняются следующие операции:

1) производится селекция буфера ввода;

2) по команде IN данные из порта ввода передаются в центральный процессор.

Модуль ввода-вывода, показанный на рис. 10.3, состоит из четырех портов вывода и двух портов ввода. Заметим,

что могут существовать и другие наборы портов ввода и вывода. Число разрядов, используемых в ЦП для записи номера порта, и определяет максимальное число адресуемых портов ввода и вывода. Если таких разрядов восемь, то максимальное число адресуемых портов ввода и вывода равно $2^8 = 256$. Это, однако, не означает, что в этом случае к микро-ЭВМ можно подключить 256 устройств ввода-вывода, так как одному УВВ часто может требоваться для работы более одного порта.

Выводы

1. Модуль ввода-вывода состоит из схемы селекции УВВ и некоторого набора портов ввода и вывода.

2. Порт вывода представляет собой регистр, в котором данные хранятся до момента их передачи устройству вывода. Содержимое порта вывода не изменяется до тех пор, пока не будет выполнена новая операция вывода информации.

3. Порт ввода представляет собой буфер, с помощью которого во время выполнения операции ввода производится выборка поданной на вход информации.

10.5. Синхронизация работы ЦП с памятью и УВВ

Как говорилось ранее, если ЦП необходимо получить какие-то данные из памяти, он посылает в память соответствующий адрес и управляющий сигнал считывания. По заданному адресу память производит селекцию нужной ячейки и передает ее содержимое в шину данных, с которой информация уже может быть принята ЦП.

Очевидно, что при получении адреса и команды считывания память не может немедленно выдать запрошенную информацию. Это выполняется в течение некоторого промежутка времени, называемого временем доступа. По истечении этого промежутка времени память посылает сигнал готовности.

До тех пор, пока ЦП не получит сигнал готовности, он должен хранить информацию. В этом случае говорят, что ЦП находится в состоянии ожидания. Ожидание ЦП завершения процесса обращения к памяти и получение сигнала готовности называют циклом ожидания готовности.

Циклы ожидания готовности могут возникнуть не только при выполнении операций типа чтения содержимого памяти, но также при организации других видов доступа ЦП к па-

мяти или УВВ, т. е. всегда, когда ЦП передает адрес и должен ждать окончания процесса декодирования, независимо от того, производится декодирование адреса в памяти или в модуле ввода-вывода. Цикл ожидания готовности является частью машинного цикла.

Существуют и другие способы использования сигналов готовности и ожидания в ЦП. Например, если сигнал готовности «привязать» к кнопке на панели управления микро-ЭВМ, то состояние ожидания можно будет продлить настолько, насколько это будет необходимо. Тогда с помощью средств индикации на панели управления можно будет исследовать состояние всех компонентов микро-ЭВМ. Каждый раз, когда нажимается кнопка на панели управления, немедленно выдается сигнал готовности. В этом случае процессор проходит через один цикл ожидания готовности, после чего он снова останавливается. Такой режим работы ЦП называют пошаговым режимом. Пошаговый режим бывает весьма полезным для контроля правильности работы микро-ЭВМ, а также при отладке программ.

10.6. Потоки данных

Структурная схема микро-ЭВМ, изображенная на рис. 10.4, построена на основе трех только что рассмотренных структурных схем модулей. Процессор, память и мо-

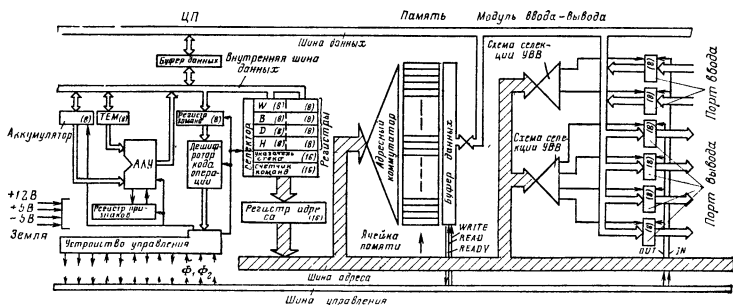


Рис. 10.4.

дули ввода-вывода подключаются друг к другу с помощью системы межмодульных связей, которая состоит из следующих компонентов:

1) 16-разрядной шины адреса, по которой ЦП передает адреса памяти к модулям ввода-вывода;

- 2) 8-разрядной шины данных;
- 3) шины управления, предназначенной для передачи таких управляющих сигналов, как «Готовность», «Считывание», «Запись» и т. д.

Структурная схема, изображенная на рис. 10.4, может служить основой для рассмотрения потоков данных, возникающих во время выполнения команд. Прежде всего рассмотрим потоки данных, которые возникают при выборке из памяти однобайтных команд. Проанализируем следующие команды.

10.6.1. Передать из регистра в регистр

С помощью этой команды организуется обмен данными между двумя регистрами общего назначения. В общем виде команда записывается как `MOV r1, r2`. Символы `MOV` являются мнемокодом операции пересылки *move*. Регистры общего назначения, участвующие в операции, символически обозначены как r_1 и r_2 .

10.6.2. Передать из памяти в регистр

Эта команда позволяет передать содержимое ячейки памяти одному из регистров общего назначения ЦП. Общая форма записи команды `MOV r, M`. Символ M здесь относится к слову памяти, адрес которого содержится в регистрах H и L .

10.6.3. Ввести данные

По этой команде данные пересылаются из порта ввода в аккумулятор. В общем виде команда записывается как `IN port`, где символы `IN` представляют собой мнемокод операции ввода *input*, а имя `port` является номером порта, т. е. адресом порта ввода, из которого передаются данные.

Выводы

1. Время доступа — это время, необходимое для декодирования адреса памяти или номера порта.

2. Ожидание центральным процессором момента, когда истечет время доступа и на его вход поступит сигнал готовности, называют циклом ожидания готовности.

3. При работе в пошаговом режиме ЦП по сигналу с панели управления выполняет один машинный цикл, после чего останавливается.

10.6.4. Выборка команды

Процесс выборки команды является частью любого цикла команды. При этом считывается из памяти и пересылается в ЦП код команды. Процесс выборки рассматриваемых

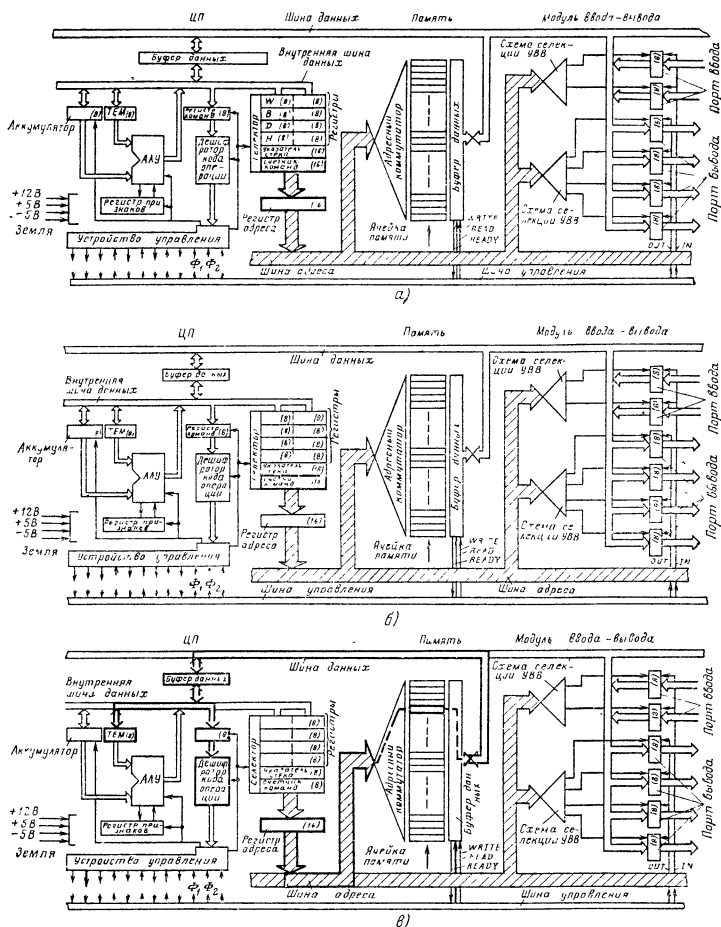


Рис. 10.5.

здесь однобайтных команд занимает один машинный цикл (М) (табл. 10.1).

Заметим, что выборка однобайтной команды проходит за три состояния. При тактовой частоте 2 МГц одно состояние

Таблица 10.1. Выборка команды

Машинный цикл	Состояние	Выполняемые действия
M1	T1	Содержимое счетчика команд передается в регистр адреса (рис. 10.5, а)
M1	T2	Содержимое счетчика команд увеличивается на 1 (рис. 10.5, б)
M1	T3	Код команды считывается из памяти и передается как в регистр команды, так и в регистр временного хранения информации ТЕМ. После этого происходит дешифрирование кода операции (рис. 10.5, в) Последующие действия полностью зависят от того, какая именно команда была выбрана из памяти

длится 500 нс. Тогда для выборки команды потребуется 3×500 нс, или 1,5 мкс.

10.6.5. Пересылка из регистра в регистр (MOV r_1, r_2)

С помощью этой команды содержимое r_2 передается в r_1 . В приведенном далее примере (табл. 10.2) в качестве r_1 и r_2 фигурируют регистры общего назначения соответственно Е и В.

Таблица 10.2. Пересылка из регистра в регистр

Машинный цикл	Состояние	Выполняемые действия
M1	T1	} Выборка команды из памяти
M1	T2	
M1	T3	
M1	T4	
		Содержимое регистра В (r_2) загружается в регистр временного хранения информации ТЕМ (рис. 10.6, а)
M1	T5	Содержимое регистра временного хранения информации ТЕМ загружается в регистр Е (r_1) (рис. 10.6, б)

Так как в этом случае машинный цикл имеет пять состояний, общее время выполнения команды будет равно 2,5 мкс (500 нс на каждое состояние).

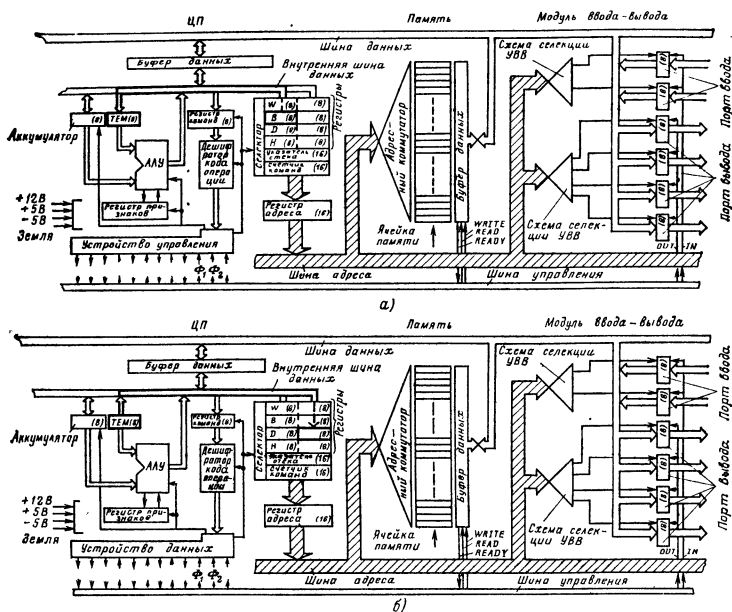


Рис. 10.6.

10.6.6. Пересылка из памяти в регистр (MOV r, M)

С помощью этой команды в регистр *r* загружается содержимое ячейки памяти *M*, адрес которой хранится в реги-

Таблица 10.3. Пересылка из памяти в регистр

Машинный цикл	Состояние	Выполняемое действие
M1	T1	Выборка команды из памяти Дешифрация команды В регистр адреса загружается содержимое пары регистров: H, L (рис. 10.7, а) Содержимое регистра адреса пересылается в память и декодируется. Информация из ячейки памяти с заданным адресом пересылается в регистр D (рис. 10.7, б)
M1	T2	
M1	T3	
M1	T4	
M2	T1	
M2	T2, T3	

страх Н и L. В данном случае (табл. 10.3) регистр г представлен регистром общего назначения D.

Машинный цикл этой команды содержит семь состояний, каждое длительностью 500 нс. При этом общее время выполнения команды равно 3,5 мкс.

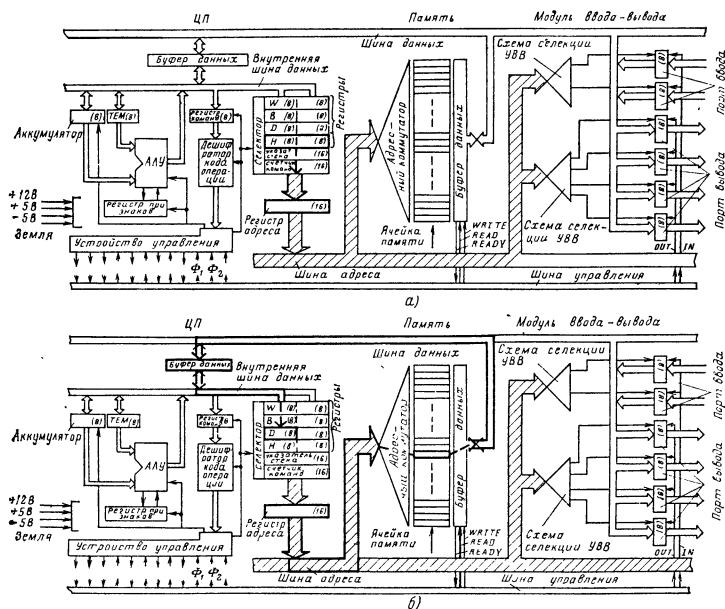


Рис. 10.7.

10.6.7. Ввод данных (IN port)

С помощью этой команды данные передаются из порта ввода в аккумулятор. Так как эта команда двухбайтная, то для ее считывания требуются два цикла обращения к памяти (табл. 10.4).

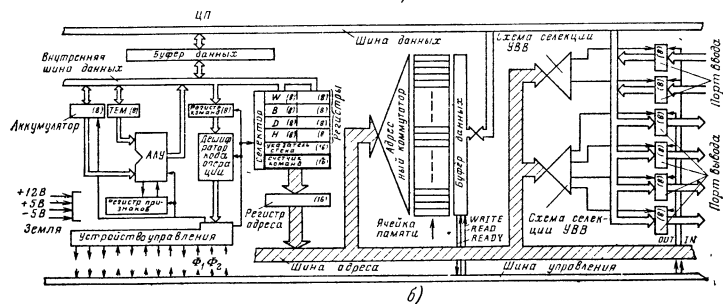
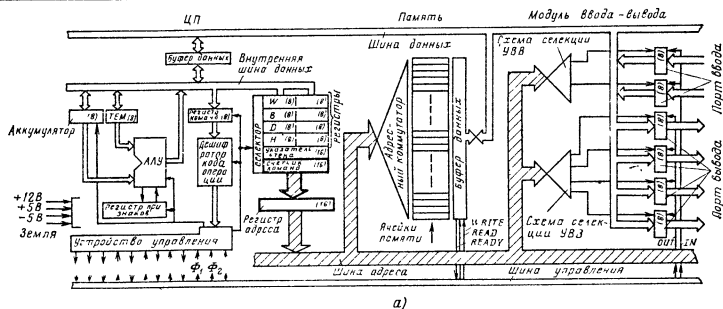
Для выполнения команды требуются десять состояний длительностью 500 нс каждое. Общее время выполнения команды при этом составит 5 мкс.

Таблица 10.4. Ввод данных

Машин- ный цикл	Состоя- ние	Выполняемое действие
M1	T1	Выборка из памяти первого байта команды (кода операции)
M1	T2	
M1	T3	
M1	T4	
M2	T1	Микро-ЭВМ выполняет некоторые вспомогательные операции Адрес второго байта команды (номера порта) передается в регистр адреса (рис. 10.8, а)
M2	T2	
M2	T3	
M2	T3	Содержимое счетчика команд увеличивается на единицу (рис. 10.8, б)
M2	T3	Второй байт команды (номер порта) считывается из памяти и загружается в регистры W и Z (рис. 10.8, в)
M3	T1	В регистр адреса загружается содержимое регистров W и Z (рис. 10.8, г)
M3	T2, T3	
M3	T2, T3	Содержимое регистра адреса передается модулю ввода-вывода, в котором происходит декодирование адреса. Содержимое порта ввода с заданным адресом пересылается в аккумулятор (рис. 10.8, д)

Выборка
команды

Выпол-
нение
команды



Глава одиннадцатая

СИСТЕМА КОМАНД

11.1. Введение

Электронной вычислительной машине, прежде чем она сможет выполнить серию последовательных операций, необходима программа. Если программа написана на языке высокого уровня, таком как PL/M, не интересны особенности конкретной микро-ЭВМ. Необходимо знать только язык.

Если записываем последовательность операций, которые необходимо выполнить на языке ассемблера, то при этом прежде всего обязаны знать, *какие* операции выполняются этой микро-ЭВМ. Другими словами, необходимо знать систему команд.

Система команд может быть разбита на пять групп:

а) команды пересылок, такие как: *ПЕРЕДАТЬ ДАННЫЕ ИЗ ПАМЯТИ В ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР*;

б) команды арифметических операций, такие как: *СЛОЖИТЬ* или *ВЫЧЕСТЬ*;

в) команды логических операций, например *ВЫПОЛНИТЬ ОПЕРАЦИЮ И НАД ДВУМЯ БАЙТАМИ*;

г) команды перехода типа JUMP (ПЕРЕЙТИ), CALL (ВЫЗВАТЬ), RETURN (ВОЗВРАТИТЬ);

д) специальные команды, например HALT (ОСТАНОВ).

Далее подробно рассмотрим некоторые команды, входящие в каждую из этих групп. Обратимся к системе команд микропроцессора Intel 8080, введенную в конце гл. 1, где приведена основная форма каждой команды. Например, буквой *г* обозначают один из регистров общего назначения (РОН) или аккумулятор в микропроцессоре 8080, а буквами *гр* одну из регистровых пар. Когда встречаются буквы *add г*, то это означает 16-битный адрес, а буквы *data* 8-битное число.

11.2. Команды пересылок

Команды пересылок предписывают МП передачу данных из одного блока в другой. Команды пересылок должны всегда определять *источник* и *приемник* данных. Для этого используются следующие правила:

а) в команде *сначала* определяют приемник, а *затем* источник данных;

б) если источник или приемник данных в команде есть ячейка памяти М, то ее адрес всегда содержится в регистрах Н и L блока РОН. Старший байт адреса всегда находится в регистре Н, а младший — в регистре L.

Применяя это правило к записи адреса $3F4A_{16}$, имеем в регистре Н $3F_{16}$, а в регистре L $4A_{16}$.

Команда MOV

В командах MOV (move) в качестве источника и приемника данных могут использоваться: регистр и ячейка памяти; ячейка памяти и регистр; регистр и другой регистр.

Пример 11.1. MOV A,B. Символ А обозначает аккумулятор, В — регистр блока РОН в микропроцессоре. Такая запись команды означает, что содержимое регистра В должно быть помещено в аккумулятор. Прежнее содержимое аккумулятора пропадает. После выполнения команды содержимое регистра В не изменяется и представляет собой просто копию содержимого аккумулятора.

Пример 11.2. MOV M,D. М — это слово в памяти, а D — один из регистров микропроцессора. Эта команда означает, что содержимое регистра D должно быть передано в ячейку памяти, адрес которой указан в регистрах Н и L.

Команды MOV однобайтные. Первые 2 бита представляют собой код операции (01), остальные 6 бит — адрес источника и приемника данных. В команде MOV M,D, например, биты 3—5 определяют, что приемником данных является ячейка памяти, адрес которой указан в регистрах Н и L. Биты 6—8 задают в качестве источника данных регистр D.

Команда MVI

Команда MVI (MoVe Immediate) отличается от команды MOV тем, что в качестве источника данных используется 8-битная константа, которая следует непосредственно за кодом операции. Приемником данных является или регистр, или ячейка памяти.

Пример 11.3. MVIM, 101111001 В. Это означает, что число 101111001_2 должно быть передано в ячейку памяти, адрес которой содержится в регистровой паре Н, L. Буква В после кода 101111001 означает, что речь идет о двоичном числе. Константа может быть также представлена в десятичной или шестнадцатеричной системе счисления. В этом случае буквы D или H нужно поместить непосредственно после

числа, записанного в команде. Ассемблирующая программа преобразует это число в двоичный код.

Пример 11.4. MVI A,D. Это означает, что буква D в коде ASCII должна быть помещена в аккумулятор. Ассемблирующая программа выполнит необходимый перевод. Буква D в кавычках означает, что D — символ. Эти кавычки не следует забывать в процессе составления программы. Если они будут присутствовать, то ЭВМ отличит символ D от чего-то другого, например от шестнадцатиричного числа.

П р и м е ч а н и е. Слово immediate означает, что этот операнд следует непосредственно за кодом операции.

Команда LDA

По команде LDA (Load Accumulator) в аккумулятор загружается содержимое ячейки памяти, адрес которой следует за кодом операции.

Пример 11.5. LDA 2FFFFH. Это означает, что содержимое ячейки памяти с адресом $2FFFF_{16}$ должно быть передано в аккумулятор. Команда LDA 3-байтная: 1 байт содержит код операции (LDA), а два других — адрес ($2FFFF_6$).

Команда LXI

Команда LXI (Load register pair Immediate) может быть использована для загрузки регистровых пар BC, DE или HL 16-битным числом (за один цикл команды). Это число непосредственно следует за кодом операции. Таким образом, команда LXI представляет собой разновидность команды MVI, которая оперирует с 8-битным числом.

Пример 11.6. Требуется загрузить в регистровую пару H,L число $3FF4_{16}$.

Р е ш е н и е 1. MVI H, 3FH
MVI L, F4H.

Р е ш е н и е 2. LXI H, 3FF4H.

Во втором случае необходимо использовать 3 байта, а в первом — 4 байта. Решение 2 лучше, так как требует меньше места в памяти.

Команды OUT, IN

Команды OUT и IN управляют обменом информацией с устройствами ввода-вывода (УВВ).

В команде OUT источником всегда является аккумулятор.

В команде IN приемником всегда является аккумулятор.

Необходимо только, чтобы *номер порта* следовал непосредственно за кодом операции IN или OUT; другими словами, в команде должен быть указан адрес УВВ, с которого должна приниматься или на который должна передаваться информация.

Пример 11.7. OUT 03H. Это означает, что содержимое аккумулятора должно быть передано в выходной порт с номером 03₁₆.

Пример 11.8. IN 06H
MOV M,A.

По первой команде данные передаются из порта ввода 06₁₆ в аккумулятор. По второй команде они засылаются в память. Адрес ячейки памяти содержится в регистровой паре HL.

П р и м е ч а н и е. В гл. 12 рассмотрены две наиболее важные команды передачи, которые называются PUSH и POP.

Вывод

1. Команды во всех системах команд могут быть разделены на команды:

- а) пересылки;
- б) арифметических операций;
- в) логических операций;
- г) передачи управления;
- д) специальные.

2. Когда в команде пересылки определяются и источник, и приемник информации, то первым после кода операции задается приемник, а затем источник.

3. Если источником или приемником информации является ячейка памяти M, то адрес этой ячейки памяти извлекается из регистровой пары H,L.

4. Обмен с внешними устройствами осуществляется по командам OUT и IN и всегда через аккумулятор.

11.3. Команды арифметических операций

Под управлением команд этой группы МП может выполнять в АЛУ арифметические операции.

Микропроцессор 8080, систему команд которого рассматриваем, может выполнять арифметические операции: сложение, вычитание.

Команда ADD

Микропроцессор 8080 представляет собой одноадресную машину. Другими словами, один из операндов всегда помещается в аккумулятор; который неявно адресуется самим кодом операции. Вслед за кодом операции необходимо указать, где находится второй операнд. Результат (сумма) помещается в аккумулятор. Операнд, который ранее находился в аккумуляторе, при этом уничтожается.

Пример 11.9. ADD H. ADD H означает, что содержимое регистра H должно быть просуммировано с содержимым аккумулятора. Сумма помещается в аккумулятор.

Пример 11.10. ADD M. Здесь содержимое ячейки памяти, адрес которой находится в регистровой паре HL, должно быть просуммировано с содержимым аккумулятора. Результат помещается в аккумулятор.

Пример 11.11. Требуется. Сложить числа 12_{10} и 84_{10} и поместить результат в ячейку памяти с адресом $2AA3_{16}$.

Решение.	MVI A, 12	Загрузить аккумулятор числом 12_{10}
	MVI B, 84	Загрузить регистр B числом 84_{10}
	ADD B	Сложить содержимое регистра B с содержимым аккумулятора и сумму поместить в аккумулятор
	LXI H, 2AA3H	Загрузить регистровую пару HI адресом $2AA3_{16}$, по которому необходимо поместить сумму
	MOV M, A	Заслать содержимое аккумулятора (сумму) в ячейку памяти с адресом $2AA3_{16}$

Команда ADC

Команда ADC (ADd with Carry) является разновидностью команды ADD. По команде ADC происходит не только сложение двух операндов, но и сложение с признаком переноса, оставшимся от предыдущей операции; результат сохраняется в аккумуляторе.

Пример 11.12. ADC M. Содержимое ячейки памяти, адрес которой записан в регистровой паре H, L, и содержимое признака переноса должно быть просуммировано с содержимым аккумулятора.

Пример 11.13. ADI 16D. Десятичное число 16 складывается с содержимым аккумулятора. Сумма помещается в аккумулятор. Если хотим сложить непосредственный опе-

ранд с содержимым аккумулятора и с признаком переноса, можно выполнить это по команде ACI (Add Immediate with Carry).

Команда INR

Команда INR (INcrement) является разновидностью команды ADD. По этой команде МП увеличивает на 1 содержимое одного из регистров блока РОН, аккумулятора или ячейки памяти.

Пример 11.14. INR C. Содержимое регистра C должно быть увеличено на 1.

Команда SUB

Команда SUB позволяет МП непосредственно вычесть содержимое одного из регистров блока РОН или содержимое ячейки памяти из содержимого аккумулятора. Другими словами, производится суммирование уменьшаемого с вычитаемым в дополнительном коде. Результат операции также помещается в аккумулятор.

Команды SBB, SUI, SBI

Команда SUB имеет следующие разновидности:
SBB = SuB tract with Borrow (вычитание с займом).
SUI = SUBtract Immediate (вычитание непосредственного операнда).
SBI = Subtract Immediate with Borrow (вычитание непосредственного операнда с займом).

Пример 11.15. SBB H. Это означает, что не только содержимое регистра H, но и значение займа вычитаются из содержимого аккумулятора. Итоговое значение займа в результате выполнения операции фиксируется в разряде переноса регистра признаков.

Пример 11.16. SUI 3AH. По этой команде вычитается шестнадцатиричное число 3A из содержимого аккумулятора. Результат операции помещается в аккумулятор. Если хотим также вычесть значение займа из содержимого аккумулятора, то нужно использовать команду SBI 3AH.

Команда DCR

Эта команда используется для уменьшения содержимого одного из регистров блока РОН, аккумулятора или ячейки памяти на 1.

Выводы

1. Каждая арифметическая операция выполняется в АЛУ микропроцессора.

2. В одноадресной машине один из операндов всегда размещается в аккумуляторе.

3. Результат выполнения арифметических операций в АЛУ фиксируется на аккумуляторе.

11.4. Команды логических операций

С помощью команд логических операций могут быть выполнены следующие действия:

а) логические операции И, ИЛИ и ИСКЛЮЧАЮЩЕЕ ИЛИ с использованием двух операндов. Один из операндов всегда размещается в аккумуляторе. Результат операции фиксируется в аккумуляторе. Таблица истинности этих операций приведена на рис. 11.1;

Операция И		
<i>x</i>	<i>y</i>	<i>f</i>
0	0	0
0	1	0
1	0	0
1	1	1

Операция ИЛИ		
<i>x</i>	<i>y</i>	<i>f</i>
0	0	0
0	1	1
1	0	1
1	1	1

Операция исключающее ИЛИ		
<i>x</i>	<i>y</i>	<i>f</i>
0	0	0
0	1	1
1	0	1
1	1	0

Рис. 11.1.

б) занесение в аккумулятор обратного кода числа;

в) сравнение двух чисел. Числа сравниваются одно с другим и результат анализируется по признакам **ЗНАК** и **НУЛЬ**;

г) сдвиг содержимого аккумулятора вправо или влево. Эта операция используется при умножении или делении чисел. Сдвиг двоичного кода числа на один разряд влево соответствует умножению числа на 2.

Команда ANA

Команда ANA (ANd with Accumalator) используется для операции логического И над содержимым одного из регистров блока РОН или ячейки памяти и содержимым аккумулятора.

Пример 11.17. ANA М. Это означает, что содержимое ячейки памяти, адрес которой находится в регистрах Н и L, участвует в операции логического И вместе с содержимым аккумулятора. Результат операции засылается в аккумулятор.

Содержимое ячейки памяти:	01110101	
Содержимое аккумулятора :	10010100	И
Результат:	00010100	

Команда ORA

Команда ORA (OR with Accumulator) используется для выполнения операции логического ИЛИ над содержимым одного из регистров блока РОН или ячейки памяти М и содержимым аккумулятора.

Пример 11.18. Произвести операцию логического ИЛИ над содержимым аккумулятора и содержимым регистра В. Результат поместить в аккумулятор.

Содержимое регистра В:	11101000	
Содержимое аккумулятора:	01001111	ИЛИ
Результат:	11101111	

Команда XRA

Команда XRA (eXclusive oR with Accumulator) используется для выполнения операции ИСКЛЮЧАЮЩЕЕ ИЛИ над содержимым одного из регистров блока РОН или ячейки памяти и содержимым аккумулятора.

Команды ANA, ORA и XRA также имеют модификации, оперирующие с непосредственным операндом. Содержимое аккумулятора участвует в операции вместе с числом, следующим за командой. Эти команды имеют следующие мнемокоды: ANI, ORI и XRI соответственно.

Пример 11.19. Требуется. Проверить состояние третьего бита (b_2) содержимого регистра В.

Решение 1. MOV A, B

ANI 00000100B

Содержимое регистра В передается в аккумулятор (все логические операции могут выполняться только там)

Выполнить логическую операцию И над содержимым аккумулятора и числом 00000100₂. Если b_2 в регистре В имеет значение 0, то в результате будет

Решение 2. MVI A, 0000C100B
ANA B

00000100₂, т. е. признак нуля не устанавливается. По содержанию признака нуля можно определить, какое значение имеет разряд b₂ регистра B: 0 или 1. Поместить в аккумулятор число 00000100₂. Выполнить над содержимым аккумулятора и содержимым регистра B операцию логического И

Теперь можно сказать, какое решение является лучшим. Оба варианта требуют три ячейки памяти.

Решение 1. MOV A, B = 1 байт
ANI 00000100B = 2 байта

Итого: 3 байта = 3 ячейки памяти

Решение 2. MVI A, 00000100B = 2 байта
ANAB = 1 байт

Итого: 3 байта = 3 ячейки памяти

Решение 2 предпочтительнее, так как оно быстрее. Время, требуемое для выполнения операции, определяется числом состояний устройства МП, которое требуется для каждой команды.

Для решения 1: MOV A, B = 5 состояний
ANI 00000100 = 7 состояний

Итого: 12 состояний

Для решения 2: MVI A, 00000100B = 7 состояний
ANA B = 4 состояния

Итого: 11 состояний

Пример 11.20. Требуется. Изменить b₅ содержимого ячейки памяти с адресом 0400₁₆. Принять содержимое этой ячейки равным 01110100₂.

Решение. LXI H, 0400H. Загрузить в регистровую пару H, L число 0400₁₆
MVI A, 00100000B Поместить число 00100000 в аккумулятор
XRA M Произвести над содержимым ячейки

памяти, адрес которой находится в регистровой паре H, L, и содержимым аккумулятора логическую операцию ИСКЛЮЧАЮЩЕЕ ИЛИ.

В результате операции изменится только содержимое разряда b₅

01	1	10100	=	Содержимое ячейки памяти с адресом 0400 ₁₆
00	1	00000	=	Число, которое используется для выполнения операции
01	0	10100	=	Результат помещается в аккумулятор

Команда CMA

Микропроцессор использует команду CMA (CoMplement Accumulator) для того, чтобы получить обратный код содержимого аккумулятора. Если хотим получить обратный код числа, находящегося в одном из регистров РОН или в ячейке памяти, то прежде всего нужно поместить это число в аккумулятор.

Пример 11.21. Т р е б у е т с я. Заменить число, содержащееся в ячейке памяти с адресом 0200₁₆, на это же число в дополнительном коде.

Решение. LXI H,0200H	Загрузить в регистровую пару H,L число 0200 ₁₆
MOV A,M	Псместить содержимое ячейки памяти в аккумулятор
CMA	Получить обратный код числа, содержащегося в аккумуляторе.
INRA	Прибавить к содержимому аккумулятора 1 (дополнительный код = обратный код + 1)
MOV M,A	Переслать содержимое аккумулятора в ячейку памяти с адресом 0200 ₁₆

Команда CMP

Как следует из мнемокода, команда CMP (CoMPare) может быть использована для сравнения двух чисел. Одно из чисел всегда помещается в аккумулятор.

При сравнении одно число вычитается из другого и результат проверяется на нуль и знак: положительный или отрицательный. Проверяется содержимое признаков переноса и нуля.

Единственная разница между командами CMP и SUB заключается в том, что при выполнении команды CMP результат операции не фиксируется в аккумуляторе. Его содержимое остается неизменным.

Разновидностью команды CMP является команда CPI (ComPare Immediate). По этой команде содержимое аккумулятора сравнивается с числом, следующим непосредственно

за кодом операции. Например, по команде CPI 14H шестнадцатиричное число 14 вычитается из содержимого аккумулятора.

Команда RLC

По команде RLC (Rotate Left in Carry) содержимое аккумулятора циклически сдвигается влево на один разряд. Другими словами, все разряды перемещаются влево на одну позицию. При этом восьмой бит перемещается в разряд признака переноса и в разряд b_0 (рис. 11.2).

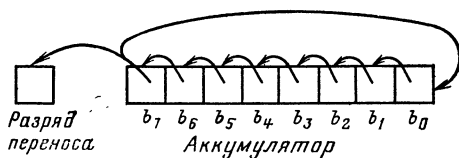


Рис. 11.2.

Если хотим выполнить операцию сдвига над содержимым ячейки памяти или содержимым одного из регистров РОН, то нужно прежде всего поместить его в аккумулятор, так как эта операция выполняется только над содержимым аккумулятора.

Пример 11.22. Т р е б у е т с я. Загрузить в триггер переноса значение разряда b_6 содержимого регистра L.

Решение. MOV A,L

RLC

RLC

Поместить содержимое регистра L в аккумулятор.

Выполнить операцию циклического сдвига влево на один разряд. При этом значение разряда b_7 переместится в разряд переноса, а b_6 займет положение b_7 .

Содержимое аккумулятора еще раз сдвигается на один разряд влево. Значение разряда b_6 помещается в разряд переноса

Команда RAL

Команда RAL (Rotate Accumulator Left) также используется для циклического сдвига влево содержимого аккумулятора на один разряд. Восьмой разряд (b_7) перемещается в разряд переноса (рис. 11.3).

Команды RRC и RAR

По команде RRC (Rotate accumulator Right in Carry) содержимое аккумулятора циклически смещается на один разряд вправо. Содержимое разряда b_0 поступает в разряд переноса и в разряд b_7 аккумулятора (рис. 11.4, а).

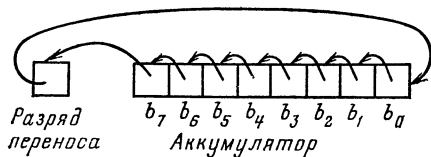


Рис. 11.3.

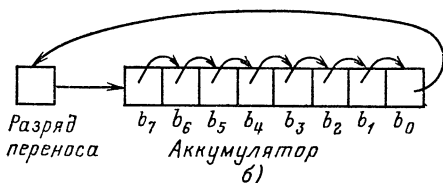
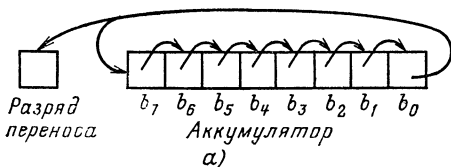


Рис. 11.4.

По команде RAR (Rotate Accumulator Right through carry) содержимое аккумулятора циклически сдвигается вправо на один разряд. Значение разряда b_0 поступает в разряд переноса. Прежнее содержимое разряда переноса поступает в разряд b_7 (рис. 11.4, б).

Выводы

1. Все логические операции МП может производить, помещая операнды в АЛУ или в процессе пересылки через АЛУ.

2. При выполнении любой логической операции один из операндов всегда находится в аккумуляторе.

3. Логическими операциями считаются не только реализация логических функций над двумя числами, но также получение дополнительного кода числа, сравнение двух чисел и циклические сдвиги содержимого аккумулятора.

11.5. Команды передачи управления

Команды располагаются в последовательных ячейках памяти, из которых они извлекаются и передаются в ЦП для выполнения в соответствии с возрастающей последовательностью адресов. Эта последовательность может быть прервана одной из команд передачи управления. Эта команда содержит в себе адрес команды, которая должна быть выполнена следующей. Команды передачи управления бывают *безусловными и условными*.

По командам безусловной передачи управления выполняется переход по программе к адресу, который указывается в команде.

По команде условной передачи управления переход по программе осуществляется только при условии, что содержимое одного из разрядов регистра признаков соответствует определенному условию.

Эти команды определяют, какое состояние разряда регистра признаков отвечает этому условию (0 или 1).

Если это условие не выполняется, переход не может быть осуществлен, и программа продолжает выполняться в соответствии с командой, следующей за командой передачи управления.

Команды условной передачи управления имеют модификации по следующим признакам регистра признаков: **ЗНАК, НУЛЬ, ПАРИТЕТ, ПЕРЕНОС**.

Команда JC

Команда JC (**J**ump **o**n **C**arry) осуществляет переход по программе только в том случае, когда в результате выполнения предыдущей операции имеет место переполнение; другими словами, если признак переноса установлен в 1.

Команда JNC

По команде JNC (**J**ump **N**o **C**arry) выполняется переход только в том случае, когда содержимое разряда переноса регистра признаков равно 0.

Пример 11.23.

BEGIN:	ADI	01H
	JNC	BEGIN
	MOV	B,A

Первые две команды увеличивают содержимое аккумулятора на 1 (ADI 01H) до тех пор, пока не появится переполнение. При отсутствии сигнала переполнения (переноса) осуществляется переход по программе к ячейке памяти с символическим адресом BEGIN. Это и есть адрес команды ADI 01H.

Когда появляется сигнал переноса, команда перехода не выполняется. Программа переходит к следующей команде (в данном случае MOV B,A).

Назначение двоеточия (:), стоящего после символического адреса BEGIN, будет пояснено в следующей главе.

Команды JZ, JNZ

Команды JZ (Jump on Zero) — команда условного перехода, которая выполняется только в случае, когда результат выполнения предыдущей операции нулевой; другими словами, если признак НУЛЬ в регистре признаков имеет значение 1.

Команда JNZ (Jump on No Zero) выполняется только при *ненулевом* результате, т. е. если признак НУЛЬ регистра признаков имеет значение 0.

Пример 11.24. Т р е б у е т с я. Загрузить в ячейку памяти 0800_{16} число 100_{10} и уменьшать его на 1 до тех пор, пока результат не станет равным 0.

Решение.	LXI H,0800H	Загрузить в регистровую пару H и L число 0800_{16}
	MVI M,100	Загрузить в ячейку памяти число 100_{10}
MIN:	DCR M	Уменьшить содержимое ячейки памяти 0800_{16} на 1
	JNZ MIN	Перейти к ячейке памяти MIN, если признак НУЛЬ регистра признаков еще не равен 0, т. е. если содержимое ячейки памяти 0800_{16} не равно 0

Команды JM, JP

Команда JM (Jump on Minus) выполняется только в том случае, когда результат выполнения предыдущей операции является *отрицательным* числом, т. е. если содержимое разряда ЗНАК регистра признаков равно 1.

Команда JP (Jump on Positive) выполняется только в том случае, когда результат выполнения предыдущей операции является положительным числом, т. е. если содержимое разряда ЗНАК регистра признаков равно 0.

Пример 11.25. Т р е б у е т с я. Вычсть содержимое ячейки памяти 0310₁₆ из содержимого ячейки памяти 0311₁₆ и перейти к команде с адресом NEG, если результат окажется отрицательным.

Р е ш е н и е. LXI H, 0311H	Загрузить в регистровую пару H и L число 0311 ₁₆
MOV A,M	Загрузить в аккумулятор содержимое ячейки памяти 0311 ₁₆
DCR L	Содержимое регистра L уменьшить на 1
SUB M	Вычсть содержимое ячейки памяти 0310 ₁₆ из содержимого аккумулятора
JM NEG	Перейти к команде с символическим адресом NEG, если результат отрицательный
—	
—	
—	

Команда JPE

Команда JPE (Jump on Parity Even) выполняется только в том случае, когда двоичный код результата предыдущей операции содержит четное число единиц.

Команда JPO

Команда JPO (Jump on Parity Odd) выполняется только в том случае, когда двоичный код результата предыдущей операции содержит нечетное число единиц.

Примеры четного результата:

01100011, 11101000, 11111111 и т. д.

Примеры нечетного результата:

11111110, 00001000, 10101000 и т. д.

Команды CALL и RET являются также командами перехода, и условия их выполнения известны.

Например, команда CNZ (Call on No Zero) выполняется только в том случае, когда признак НУЛЬ в регистре признаков не равен 1 в данный момент. В свою очередь, команда RC (Return on Carry) выполняется только при наличии признака переноса.

Команды CALL и RET подробно рассмотрены в гл. 12.

11.6. Специальные команды

Каждый МП имеет ряд специальных команд. Эти команды не передают и не обрабатывают информацию, но они используются для управления работой МП.

Команда HLT

По команде HLT (Halt) останавливается текущая программа до тех пор, пока не появится запрос прерывания от устройства ввода-вывода.

Команды DI, EI

Если МП получает команду DI (Disable Interrupt), то он игнорирует запросы прерывания до тех пор, пока не поступит команда EI (Enable Interrupt).

Пример 11.26.

DI Запросы прерывания больше не принимаются.

HLT Микропроцессор остановлен до тех пор, пока не поступит запрос прерывания.

Микропроцессор может быть вновь пущен только сигналом сброса RESET. При этом счетчик команд обнуляется, и выполнение программы начинается сначала.

11.7. Пример программы

Имеется произвольный ряд чисел, хранимых в ячейках памяти с последовательными адресами. Первое число расположено в памяти по адресу 0930.

Т р е б у е т с я. Записать программу, по которой эти числа будут складываться друг с другом до появления признака переполнения. Адрес последнего числа, участвующего в сложении, должен быть выдан в порты 04 и 05.

```
Решение. LXI    H,0930H
              MOV    A,M
AGAIN: INR
              ADDM
              JNC AGAIN
              MOV    A,H
              OUT    04H
              MOV    A,L
              OUT    05H
              HLT
```

По команде LXI H,0930H в регистровую пару H,L загружается адрес первого числа, участвующего в сложении. По команде MOV A,M это число перемещается в аккумулятор. По команде INR L содержимое регистра L увеличивается на 1 (в регистровую пару H,L поступает число 0931H, адрес второго слагаемого). Команда ADD M прибавляет это число к содержимому аккумулятора. Теперь число, находящееся в аккумуляторе, есть сумма двух первых чисел.

По команде JNC AGAIN осуществляется проверка содержимого разряда переноса регистра признаков. Если переполнения нет, то будет выполняться команда, находящаяся по символическому адресу AGAIN, т. е. команда INR L.

Этот цикл программы выполняется до тех пор, пока не появляется признак переноса. В этом случае команда JNC передает управление следующей по программе команде. Адрес последнего числа, участвовавшего в сложении, зафиксирован в регистровой паре H,L и должен быть передан в выходные порты.

Так как команда OUT оперирует только с содержимым аккумулятора, то содержимое регистровой пары H,L должно быть сначала занесено в аккумулятор. По команде MOV A,H содержимое регистра H переносится в аккумулятор, а по команде OUT 04H помещается в выходной порт 04H. Команды MOV A,L и OUT 05H передают содержимое регистра L через аккумулятор в выходной порт 05H. Затем микро-ЭВМ останавливается по команде HLT. В результате, если предположить, что первый байт первой команды этой программы находился в памяти программ в ячейке с адресом 2000, вся трехбайтная команда LXI H,0930H займет ячейки памяти с адресами 2000, 2001, 2002. Однобайтная команда MOV A,M располагается по адресу 2003. Команда INR хранится в ячейке памяти с адресом 2004. Таким образом, символический адрес AGAIN имеет действительное значение 2004.

Выводы

1. Команды передачи управления делятся на условные и безусловные.

2. В условных командах передачи управления переход выполняется в зависимости от значения одного из признаков: ПЕРЕНОС, ЗНАК, НУЛЬ и ПАРИТЕТ.

3. Если в команде условной передачи управления не выполняется заданное условие, то переход по программе не имеет места и выполняется следующая команда программы.

Глава двенадцатая СИНТАКСИС И ПОДПРОГРАММЫ

12.1. Введение

В гл. 1 были объяснены причины использования языка ассемблера при написании программ для микро-ЭВМ вместо двоичных кодов. Так как язык ассемблера ориентирован на конкретную микро-ЭВМ, отличную от машин подобного класса, то и языки ассемблера будут различными. Эти различия в основном относятся к терминам, используемым в командах, и способам адресации операндов. Язык ассемблера должен соответствовать специфическим особенностям МП. Имеется большая разница между одно- и двух-адресной машинами. Важным фактором является следующий: имеет машина регистры общего назначения или нет.

Различие между языками ассемблера из-за разнообразия микро-ЭВМ приводит к следующему:

1. Программу необходимо писать на языке ассемблера для того типа МП, для которого она предназначена.

2. Когда исходную программу, написанную на языке ассемблера, преобразуют в объективную программу, используемая программа трансляции должна соответствовать этому типу МП. Это *не* обязательно означает, что трансляция должна выполняться на этом конкретном МП. Она может быть также выполнена на большой ЭВМ, имеющей в составе программного обеспечения эту ассемблирующую программу. В этом случае ассемблирующая программа записана на языке большой ЭВМ, но она оперирует с исходной программой, записанной на языке ассемблера микро-ЭВМ. Объектная программа также выдается на языке микро-ЭВМ.

В первой части этой главы рассматриваются вопросы, общие для всех языков ассемблера, — *синтаксис*. Это — общая конструкция и метод записи команд ассемблера. Вторая часть главы является по существу продолжением гл. 11, однако здесь в основном рассматривается механизм *подпрограмм*.

12.2. Структура команд ассемблера

Хотя мнемокоды одинаковых команд часто различны для разных микро-ЭВМ, конструкция записи (часто называемая синтаксисом) команд ассемблера одинакова для всех языков ассемблерного уровня.

Команда, написанная на языке ассемблера, может быть разделена на четыре части (часто называемые полями):

МЕТКА ОПЕРАЦИЯ ОПЕРАНД КОММЕНТАРИЙ

Они имеют следующее значение.

Операция. В поле операции команды для записи кода операции применяется буквенный код. Этот буквенный код является обычно аббревиатурой полного названия операции.

Пример.

ADD — СЛОЖИТЬ
SUB — ВЫЧЕСТЬ
JMP — ПЕРЕЙТИ
JNC — ПЕРЕЙТИ ПРИ ОТСУТСТВИИ ПЕРЕНОСА
RRC — ЦИКЛИЧЕСКИ СДВИНУТЬ ВПРАВО
 ЧЕРЕЗ ТРИГГЕР ПЕРЕНОСА

Операнд. Поле операнда команды ассемблера используется для определения данных (операндов), которые должны участвовать в выполнении данной операции. В зависимости от типа команды поле операнда может содержать:

	ОПЕРАЦИЯ	ОПЕРАНД
Операнд, например:	ANI	35H
Адрес операнда, например:	LDA	03FFH
Адрес операнда и операнд, например:	MVI	A,11000101B
Адреса двух операндов, например:	MOV	A,B

Если в поле операнда заданы два операнда, то они должны разделяться запятой.

Метка. При написании программы необходимо специфицировать адреса ячеек памяти, в которых размещены коды операций. Это делается, как показано на рис. 12.1.

Предположим, что имеется часть программы, состоящей из пяти команд форматом в 1, 2 или 3 байта. Последовательность байт размещена по адресам памяти от 01₁₀ до 12₁₀. Пятая команда — команда перехода. Адрес перехода дан

после мнемокода операции JMR. Способ, с помощью которого вводится программа в микро-ЭВМ, определяет, нужно

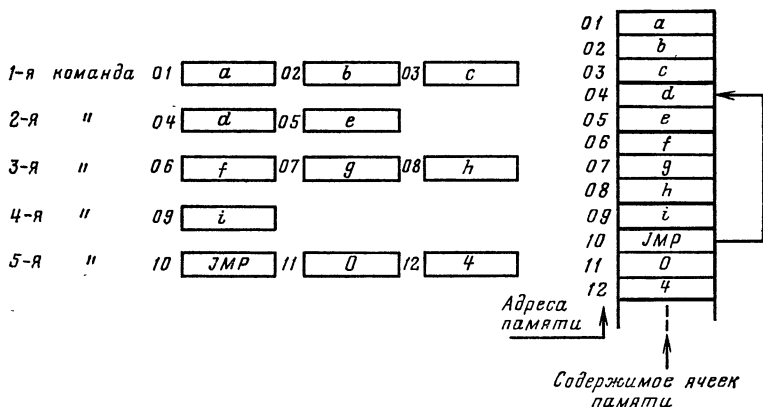


Рис. 12.1.

ли самим вычислять эти адреса или это делается автоматически в процессе трансляции.

Ввод программы с пульта управления

В этом случае адреса определяем самостоятельно. Затем вводим адреса и команды в двоичных кодах с помощью переключателей. Адресные переключатели (обычно 16) и переключатели данных (обычно 8) ставятся в требуемое положение (0 или 1), и нажимается кнопка, с помощью которой параллельным кодом вводятся данные в выбранную ячейку памяти. Так как можно вводить числа только в двоичной системе, то необходимо перевести мнемокод команды также в двоичный код операции. Все адреса регистров и шестнадцатиричные числа (например, адреса памяти) должны быть преобразованы в свои эквиваленты в двоичной системе счисления. Фактически берем на себя функции ассемблирующей программы.

Ввод с использованием ассемблирующих программ

Действительные адреса определять не нужно. Ассемблирующая программа вычисляет адреса и ставит им в соответствие последовательности команд. Необходимо, однако,

указать ассемблеру адрес первого байта первой команды программы. Так как сами не определяем адреса, ~~можно~~ задавать в команде символический адрес. Команда перехода всегда содержит адрес. Необходимо, чтобы адрес, по которому должен быть выполнен переход, был ясно указан. Это обеспечивается присвоением адресу перехода собственного символического имени, или *метки*. Метка записывается в *поле метки*. Таким образом, метка не число, а имя. Метка ставится после мнемокода операции перехода. Ассемблирующая программа в процессе трансляции заменяет метку соответствующим адресом.

Пример.

МЕТКА	ОПЕРАЦИЯ	ОПЕРАНД
FORWARD:	MOV	A,B
	ADD	
	—	
	—	
	JMP	FORWARD

Метка FORWARD (ВПЕРЕД) связана с командой MOV (ПЕРЕДАТЬ ОПЕРАНД ИЗ В в А). Ассемблер следит за адресацией при введении программы. Если ассемблер встретит команду JMP FORWARD, то он подставит действительное значение адреса вместо символического имени FORWARD в команду перехода.

Комментарий

Программист может использовать это поле для пояснения команды. Поле комментария используется только для удобства чтения программы и полностью игнорируется ассемблирующей программой в процессе трансляции. Если программа отпечатана на телетайпе, снабженном перфоратором, то каждый символ, который печатается, преобразуется на перфоленде в код ASCII. Эта лента называется *исходной* и содержит также поле комментария.

Выводы

1. Общая структура команд ассемблера является следующей:

МЕТКА ОПЕРАЦИЯ ОПЕРАНД КОММЕНТАРИЙ

2. Поле операции содержит мнемонический код операции.

3. Поле операнда используется для задания объекта операции.

4. Поле метки используется программистом для символического представления адреса ячейки памяти, по которому должен быть осуществлен переход.

5. Поле комментария дает программисту дополнительную информацию по данной команде ассемблера.

12.3. Печать команд ассемблера

Телетайп является устройством, которое чаще всего используется для ввода программ, написанных на языке ассемблера.

Телетайп может использоваться в качестве клавиатуры, печатающего механизма, перфоратора, устройства считывания с перфоленты.

Для преобразования написанной от руки программы в программу, которая хранится в памяти микро-ЭВМ, требуется выполнить следующие действия (рис. 12.2):

1. Включить телетайп.

2. Запустить на микро-ЭВМ ассемблирующую программу трансляции.

При этом возможны два случая:

а) программа трансляции хранится в ПЗУ микро-ЭВМ;
б) программа трансляции хранится на гибком диске, магнитной ленте или перфоленте. В этом случае микро-ЭВМ должна быть соединена с оборудованием, которое ведет программу трансляции.

3. Печатать на телетайпе программу строку за строкой в соответствии с правилами записи программ для конкретного МП. Телетайп при этом преобразует числа и буквы в символы, закодированные в системе ASCII, и вводит их в память микро-ЭВМ в виде электрических сигналов.

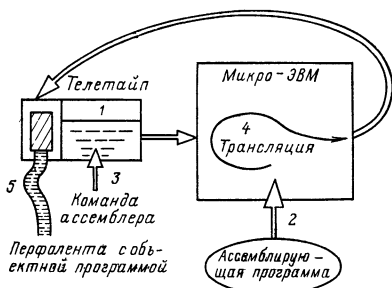


Рис. 12.2.

Когда программа напечатана на телетайпе, то ее проверяют на наличие ошибок.

Если это опечатки, то их можно исправить с помощью вспомогательной программы-редактора.

4. Преобразовать в микро-ЭВМ исходную программу в объектную. Процесс трансляции управляется ассемблирующей программой. Объектная программа загружается в память микро-ЭВМ.

5. С помощью телетайпа вывести из памяти объектную программу на перфоленту для хранения.

LABEL	MNEMONIC	OPERAND	COMMENT
	ORG	1100H	
ZERO	EQU	03H	
STORE	EQU	1200H	
CI	EQU	1009H	
VUL:	LXI	H,STORE	;INITIALISATE POINTER
READ:	CALL	CI	;CHARACTER IN FROM CONSOLE
	MOV	M,A	;STORE CHARACTER
	CPI	ZERO	
	RZ		;RETURN IF ZERO
	INX	H	;INCREMENT POINTER
	JMP	READ	
	END		

Рис. 12.3.

Имеются и другие методы преобразования исходной программы в объектную. Они рассмотрены далее в гл. 18 и 19.

При печати команд ассемблера на телетайпе необходимо придерживаться следующих правил:

а) для каждой команды должна резервироваться одна строка. Необходимо нажимать на клавиши «Новая строка» и «Возврат каретки» в конце каждой команды, в результате чего печатание следующей команды начнется с новой строки (если телетайп соединен с перфоратором, то на ленте пробивается специальная комбинация отверстий, указывающая конец строки);

б) поля команды ассемблера разделяются одним или несколькими пробелами. Эти пробелы должны выполняться так, чтобы соответствующие поля команд всегда находились одно над другим и более четко выделялись в тексте программы;

в) если метка отсутствует, то первое поле оставляется пустым;

г) в некоторых ассемблерах необходимо после метки поставить двоеточие. В других ассемблерах достаточно оставить пробел между полями метки и операции.

Метка может состоять из любой комбинации букв, чисел или знаков. В некоторых ассемблирующих программах накладываются определенные ограничения на способ записи метки; так, например, метка не может начинаться с цифры или не должна быть такой же, как команда ассемблера;

д) в некоторых ассемблирующих программах поле комментария должно начинаться с точки с запятой, в других программах достаточно пробела между полями операнда и комментария.

На рис. 12.3 приводится пример нескольких строк программы на языке ассемблера.

12.4. Директивы ассемблера

Ассемблирующая программа (ассемблер) переводит исходные программы, записанные на языке ассемблера, в объектные, которые может читать микро-ЭВМ.

Хотя ассемблирующая программа в процессе трансляции берет на себя многие функции программиста, такие как адресация, преобразование чисел и др., программист все-таки должен давать ассемблеру некоторые уточнения. Например, ассемблер не знает, в какую ячейку памяти должна быть помещена первая команда программы. Без специальных указаний ассемблер не может также установить конец программы.

Программист должен выдать эту информацию ассемблирующей программе в *директивах ассемблера*.

Директива ассемблера не отображается машинным кодом в объектной программе, а является просто указанием ассемблеру.

Ниже приводится несколько примеров директив ассемблера:

МЕТКА	ОПЕРАЦИЯ	ОПЕРАНД	КОММЕНТАРИЙ
	ORG	0400H	
CONS	EQU	05H	
	END		

Директива ORG (origin — начало) задает ассемблеру адрес ячейки памяти для первой команды транслируемой программы.

Директива ORG 0400H обеспечивает размещение по адресу 0400₁₆ первой команды программы.

Директива ассемблера EQU (equate — приравнять) используется, когда при написании программы некоторому символическому имени ставится в соответствие определенный операнд.

Директива EQU присваивает определенное значение этому символическому имени. Например:

МЕТКА	ОПЕРАЦИЯ	ОПЕРАНД
CONS	EQU	05H
	MVI	A,CONS

Операнд CONS заменяется значением 05₁₆ в процессе ассемблирования.

Директива ассемблера END — конец — используется для указания того, что исходная программа закончилась.

П р и м е ч а н и е. Когда в программе ставится в соответствие символическое имя и адрес ячейки памяти, не нужно сообщать ассемблеру действительное значение этого адреса с помощью директивы EQU. Ассемблер делает это сам, используя *счетчик ячеек*, который является составной частью ассемблирующей программы. Директивой EQU сообщается ассемблеру адрес первой команды программы. После трансляции каждой команды содержимое счетчика ячеек увеличивается на число, которое соответствует формату команды в байтах.

Выводы

1. Телетайп — это устройство, которое чаще всего используется в микро-ЭВМ для ввода программы на языке ассемблера.

2. Для каждой команды ассемблера отводится одна строка.

Поля команды ассемблера отделяются друг от друга пробелами.

В некоторых случаях поле метки заканчивается двоеточием.

В некоторых случаях поле комментария начинается с точки с запятой.

3. Директивы ассемблера — это указания, которые задает программист ассемблирующей программе.

4. Директива ассемблера не отображается в объектной программе.

12.5. Подпрограммы

Довольно часто одна и та же группа команд, реализующих некоторую вычислительную процедуру, должна повторяться в различных местах программы. Очевидно, что если эти группы команд вводить в программу там, где они должны выполняться, то это будет расточительным использованием емкости памяти, особенно если они встречаются в программе часто. В этом случае лучше записать эти группы команд как подпрограммы и разместить их где-то в памяти отдельно от главной программы (рис. 12.4). Подпрограмму можно вызывать из главной программы по мере необходимости с использованием специальной команды. Рассмотрим команды, которые применяются при работе с подпрограммами.

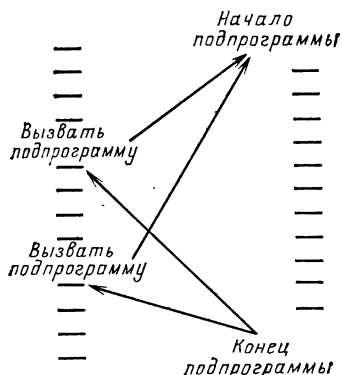


Рис. 12.4.

12.6. Команды вызова подпрограмм

Команда CALL (вызвать) вызывает подпрограмму. Она состоит из 3 байт: 1 байт — для кода операции (CALL) и 2 байта для указания адреса первой команды подпрограммы. Этот адрес обычно задается символическим именем, например:

ОПЕРАЦИЯ	ОПЕРАНД
CALL	MULTI

Кроме того, по команде выполняются действия, необходимые для возвращения в основную программу после выполнения подпрограммы:

а) в счетчике команд фиксируется адрес команды в основной программе, которая следует за командой вызова;

б) содержимое счетчика команд (адрес возврата) загружается в стек; содержимое указателя стека модифицируется;

в) в счетчик команд загружается адрес, задаваемый командой вызова.

После этого может начаться исполнение подпрограммы.

12.7. Команды возврата из подпрограмм

Команда RET (return — возврат) является последней командой подпрограммы. По этой команде выполняется возврат к главной программе, подготовленный командой вызова. Команда возврата содержит только код операции. По этой команде происходит следующее:

- а) счетчик команд получает из стека адрес команды в основной программе, следующей за командой вызова;
- б) содержимое указателя стека соответственно модифицируется.

12.8. Команды управления стеком

Часто бывает так, что при выполнении подпрограммы требуется использовать один или несколько регистров общего назначения ЦП. Однако эти регистры могут содержать данные, которые потребуются позднее в основной программе. Поэтому эти данные необходимо поместить в стек до начала выполнения подпрограммы. Это делается по команде PUSH (поместить в стек). Команда POP (вытолкнуть из стека) используется для возврата данных в регистры после выполнения подпрограммы. Команды PUSH и POP всегда оперируют с парой регистров. Такими парами регистров могут быть: BC, DE, HL; аккумулятор и регистр признаков.

Содержимое аккумулятора и регистра признаков вместе образует так называемое слово состояния программы PSW (Program Status Word).

Слова в стеке являются 8-разрядными. Поэтому чтобы поместить в стек содержимое пары регистров (16 разрядов), требуются два этапа; при этом указатель стека указывает на последний адрес в стеке, по которому были помещены данные (в некоторых МП, однако, указатель стека показывает первую свободную ячейку в стеке). По команде ПОМЕСТИТЬ В СТЕК выполняются следующие действия:

- а) содержимое указателя стека сначала декрементируется (уменьшается на 1);
- б) первые 8 бит загружаются в стек;
- в) содержимое указателя стека вновь декрементируется;
- г) последующие 8 бит загружаются в стек.

По команде ВЫТОЛКНУТЬ ИЗ СТЕКА выполняются обратные действия. Сначала второй байт выводится из стека в ЦП, и содержимое указателя стека инкременти-

руется. Затем первый байт передается из стекла в ЦП и указатель стека вновь инкрементируется.

Очевидно, что до начала выполнения рабочей программы необходимо дать информацию микро-ЭВМ относительно области памяти, которая резервируется для стека. Другими словами, сначала необходимо загрузить в указатель стека наибольший адрес стека плюс 1 (указатель стека декрементируется перед загрузкой данных в стек). Это можно проиллюстрировать следующим примером.

Т а б л и ц а 12.1

Программа	Подпрограмма	Комментарий
—	MULTI: PUSH PSW	; Поместить в стек содержимое регистра признаков, аккумулятора и регистров D, E, H и L
—	PUSH D	
—	PUSH H	
—	—	
—	—	
0100 CALL MULTI	POP H	; Вытолкнуть из стека в ЦП содержимое регистров H, L, D, E, аккумулятора и регистра признаков
0103 —	POP D	
—	POP PSW	
—	RET	; Загрузить в счетчик команд адрес возврата 0103 ₁₆

12.9. Пример

Фрагмент программы показан слева в табл. 12.1. В этой программе в некоторый момент вызывается подпрограмма с использованием команды CALL MULTI.

Команда CALL MULTI трехбайтная: 1 байт задает операции и 2 байта используются для указания адреса первой команды подпрограммы. Этому адресу дали символическое имя MULTI.

Часто символическое имя для подпрограммы выбирают таким образом, чтобы можно было понять, какие действия выполняет сама подпрограмма. Имя MULTI выбрали потому, что это подпрограмма для выполнения умножения.

Команда CALL MULTI занимает три ячейки памяти. Код операции (CALL — вызвать) находится по адресу

0100₁₆ (табл. 12.1). Два байта адреса MULTI первой команды подпрограммы занимают ячейки памяти с адресами 0101₁₆ и 0102₁₆ соответственно. Поэтому следующая команда в основной программе находится по адресу 0103₁₆. Отсюда следует, что именно этот адрес зафиксирован в счетчике команд после того, как осуществлена выборка команды CALL MULTI.

До того, как в счетчик команд будет загружен адрес MULTI, его прежнее содержимое должно быть перенесено в стек (0103₁₆). Это делается по команде CALL, которая декрементирует указатель стека и загружает в стек первый байт содержимого счетчика команд. Затем указатель стека снова декрементируется и в стек переносится второй байт.

На рис. 12.5, а показано содержимое стека. Ячейки памяти с 0500₁₆ по 05FF₁₆ зарезервированы для стека. Когда загружаются данные в стек, необходимо не выйти за пределы отведенной области адресов памяти, так как в противном случае можно испортить рабочую программу, также размещенную в памяти.

Перед тем, как начать выполнение подпрограммы, сначала необходимо поместить в стек содержимое аккумулятора, регистра признаков и регистров D, E, H, L. Это выполняется с помощью команд PUSH PSW, PUSH D и PUSH H (табл. 12.1). Так как команды PUSH всегда оперируют с парами регистров, то в командах указываются только первые регистры пар (PUSH H означает PUSH H и L).

На рис. 12.5, б показано содержимое стека после выполнения трех команд PUSH. На этом рисунке (A) — содержимое аккумулятора; (Признаки) — содержимое регистра признаков и т. д.

Только после загрузки содержимого регистров в стек может выполняться подпрограмма. Когда подпрограмма выполнена, содержимое регистров D, E, H, L, регистра признаков и аккумулятора должно быть восстановлено в ЦП.

Так как указатель стека непрерывно декрементируется при загрузке данных в стек (PUSH) и инкрементируется при выборке данных (POP), байт, загруженный последним, будет выдан первым. Этот принцип адресации памяти часто называется LIFO (Last-In-First-Out).

Последовательность команд POP (вытолкнуть из стека) должна быть обратной последовательности команд PUSH (поместить в стек). Если программист забудет об этом, то

может, например, получиться так, что данные, которые находились в паре регистров H, L до выполнения подпро-

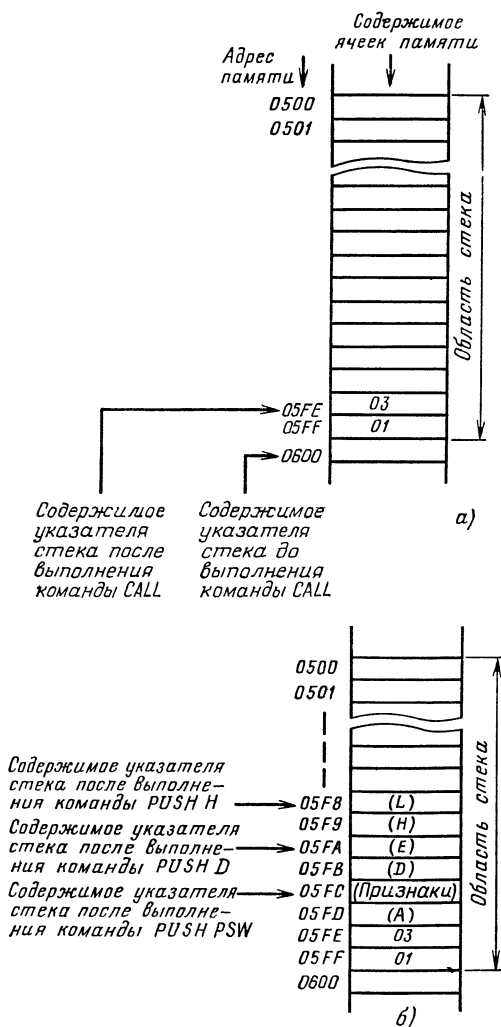


Рис. 12.5.

граммы, окажутся в паре регистров D, E после ее выполнения. После того, как данные из стека будут восстановлены в регистрах ЦП, используется команда RET (вернуться),

по которой адрес возврата 0103_{16} из стека помещается в счетчик команд, и продолжается выполнение основной программы.

Глава тринадцатая ТЕХНИКА АДРЕСАЦИИ

13.1. Введение

В данной главе описываются способы адресации, с помощью которых определяются операнды или адреса операндов в командах.

Самым важным при программировании является выбор способов адресации, которые используют минимальный объем памяти для хранения программы и наименьшее время ее исполнения.

Так как техника адресации редко удовлетворяет обоим условиям, обычно нужно искать компромиссное решение. Выбор техники адресации, конечно, определяется также набором команд, которым располагает определенный МП.

13.2. Непосредственная адресация

При непосредственной адресации операнд следует сразу за кодом операции (рис. 13.1). Когда код операции введен в ЦП, то счетчик команд указывает на операнд.

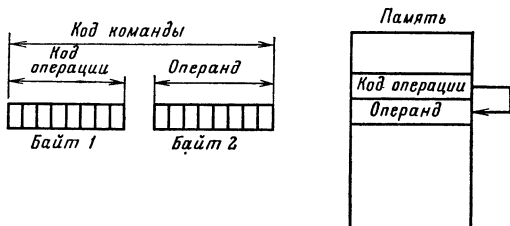


Рис. 13.1.

Так как за кодом операции следует сам операнд, а не адрес операнда, адресации в настоящем смысле слова не происходит. Операнды хранятся в памяти в ячейке, которая следует непосредственно за ячейкой, содержащей код операции. Многие команды непосредственной адресации яв-

ляются 2-байтными. Первый байт содержит код операции, а второй — операнд. Непосредственная адресация применяется, когда операнд известен при написании программы. Непосредственная адресация обозначается символом I (Immediate). Например:

MVI A, 01101010B	Передать непосредственный операнд в аккумулятор	Загрузить непосредственный операнд в аккумулятор
ADI 01101010B	Сложить непосредственный операнд с содержимым аккумулятора	Сложить непосредственный операнд с содержимым аккумулятора
ANI 01101010B	Логическое И непосредственного операнда и аккумулятора	Выполнить операцию логического И над содержимым аккумулятора и непосредственным операндом

13.3. Прямая адресация

О прямой адресации говорят, когда адрес данных, участвующих в операции, следует за кодом операции. Этим адресом может быть: адрес памяти; имя регистра; номер порта.

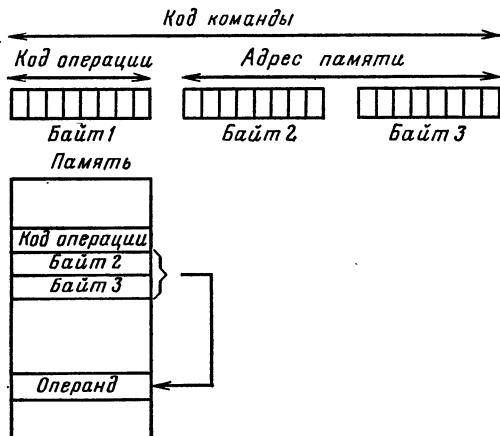


Рис. 13.2.

Адрес памяти. Команды этого типа являются 3-байтными, так как память адресуется двумя байтами (рис. 13.2). Например:

LDA 2FFFH	Загрузить аккумулятор с прямой адресацией	Загрузить аккумулятор содержимым ячейки памяти с адресом 2FFF ₁₆
STA 0300H	Запомнить содержимое аккумулятора с прямой адресацией	Передать содержимое аккумулятора в ячейку памяти с адресом 0300 ₁₆

Имя регистра. Команды этого типа являются 1-байтными, так как код операции 2-разрядный, а адрес регистра 3-разрядный. При необходимости в такой команде могут

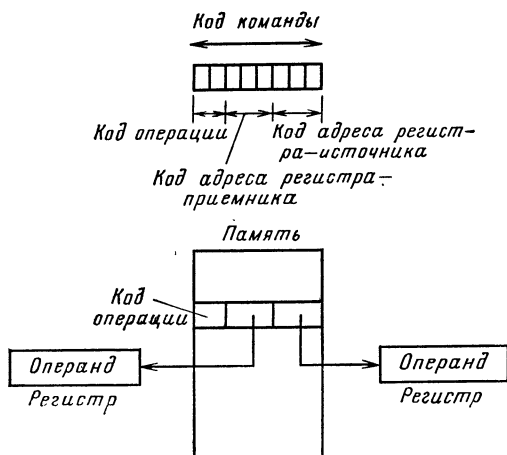


Рис. 13.3.

содержаться два адреса регистров; это будет в том случае, когда данные передаются с одного регистра на другой (рис. 3.3). Например:

INR B	Инкрементировать содержимое регистра	Увеличить содержимое регистра В на 1
MOV A,B	Передать содержимое регистра в регистр	Переместить содержимое регистра В в аккумулятор

Номер порта. Команды этого типа 2-байтные, так как требуется восемь разрядов (1 байт) для задания адреса порта (рис. 13.4). Например:

IN 06H	Ввод	Ввести содержимое порта ввода с номером 06 ₁₆ в аккумулятор
OUT 12H	Вывод	Выдать содержимое аккумулятора в порт вывода с номером 12 ₁₆

Примечание. В одной и той же команде могут применяться прямая и непосредственная адресации. С непосредственной адресацией часто может применяться и косвенная адресация, которая будет рассмотрена далее.

При этом форматы команд будут различными. Ниже приводятся примеры использования комбинации прямой и непосредственной адресации:

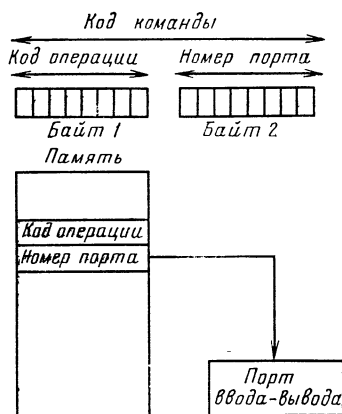


Рис. 13.4.

MVI B, 12H

Передать непосредственный операнд в регистр

Загрузить в регистр В непосредственный операнд 12_{16}

LXI H, 2A49H

Загрузить регистровую пару непосредственным операндом

Загрузить в пару регистров H и L непосредственный операнд $2A49_{16}$

Вывод 1

1. При непосредственной адресации операнд следует непосредственно за кодом операции.

2. В командах, использующих прямую адресацию, адрес операнда следует сразу за кодом операции. Адресами операнда могут быть адрес ячейки памяти, имя регистра или номер порта.

13.4. Косвенная адресация

В этом методе адресации используются фактически два адреса. Первый адрес включен в команду и обычно является символическим именем регистра. Однако в этом регистре (или паре регистров) нет операнда, как при прямой адресации, а содержится адрес ячейки памяти. Операнд находится поэтому в памяти (рис. 13.5). Так как для адресации памяти требуется 16 разрядов, адрес определяется содержанием двух 8-разрядных регистров (парой регистров). В коде команды задается адрес только первого регистра (старшего), так как в коде операции уже указано, что используется

пара регистров. Формат команды зависит от того, каким образом адресуется второй операнд. Например, если второй операнд размещен в регистре, то для команды требуется 1 байт (рис. 13.5).

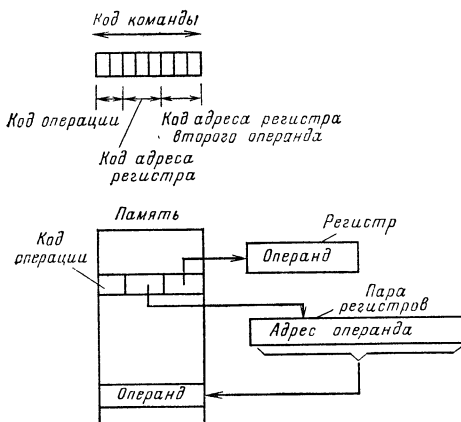


Рис. 13.5.

Примеры.

ADD M	Сложить содержимое памяти с содержимым аккумулятора	Содержимое ячейки памяти, адрес которой находится в паре регистров H, L, суммируется с содержимым аккумулятора
INR M	Инкрементировать память	Добавить единицу к содержимому ячейки памяти, адрес которой находится в паре регистров H, L

Примеры сочетания метода косвенной адресации с другими методами адресации:

MOV B, M	Передать содержимое памяти в регистр B	Передать содержимое ячейки, адрес которой находится в паре регистров H, L, в регистр B
MUI M, 2AH	Передать непосредственный операнд в память	Загрузить в ячейку памяти, адрес которой находится в паре регистров H, L, непосредственный операнд 2A ₁₆

13.5. Неявная адресация

При этом методе адресации, известном также как подразумеваемая адресация, адрес одного или обоих операндов неявно задается в коде операции. Операция может, например, выполняться только с содержимым аккумулятора. В этом случае аккумулятор уже адресуется кодом команды и нет необходимости его упоминать специально. Именно так построены все логические и арифметические команды в МП модели 8080. Первый операнд всегда находится в аккумуляторе. Второй операнд может быть адресован прямо, косвенно или непосредственно. От способа адресации зависит формат команды.

Примеры.

ANA B	Логическое И содержимого регистра с содержимым аккумулятора	Выполнить операцию логического И над содержимым аккумулятора и содержимым регистра B
ADD M	Сложить содержимое памяти с содержимым аккумулятора	Суммировать содержимое ячейки памяти, адрес которой задан в регистрах H и L, с содержимым аккумулятора

Выводы

1. При косвенной адресации адрес операнда находится в паре регистров.

Имя первого (старшего) регистра задается непосредственно после кода операции. Так как из кода операции следует, что имеет место косвенная адресация, второй (младший) регистр определяется автоматически.

2. При неявной адресации адрес операнда неявно задается в коде операции. Один из операндов такой команды, например, всегда находится в аккумуляторе.

13.6. Относительная адресация

Относительная адресация — это свободное название ряда методов, имеющих один общий принцип. При относительной адресации адрес памяти операнда равен сумме базового значения адреса и «смещения». Понятие «смещение» иллюстрируется на рис. 13.6. Предположим, что в памяти, как показано на рис. 13.6, содержится таблица, и что каждая ее строка 2-байтная. Адрес первого байта b первой строки рассматривается как базовый. Смещение — это

число (число байт), которое должно быть добавлено к адресу b , чтобы найти требуемую строку. В рассматриваемом случае это число 2 для второй строки, 4 — для третьей и т. д. Рассмотрим отдельные виды относительной адресации.

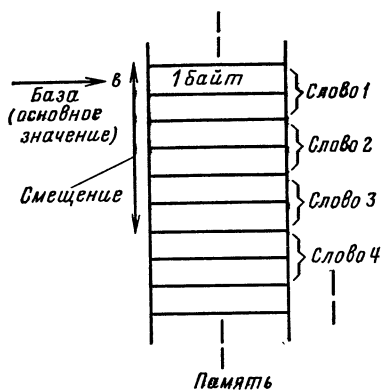


Рис. 13.6.

Постраничная адресация. В некоторых микро-ЭВМ используется так называемый постраничный способ (метод) адресации памяти. Это означает, что память подразделяется на блоки, в каждом из которых содержится определенное число машинных слов. Такой блок называется страницей. Блок может быть организован таким образом, что первый байт

адреса будет одним и тем же для всех слов памяти на странице. Когда код команды и ее операнд находятся на одной странице, первый байт содержимого счетчика команд может быть использован в качестве первого байта адреса памяти. Таким образом, в команде достаточно ука-

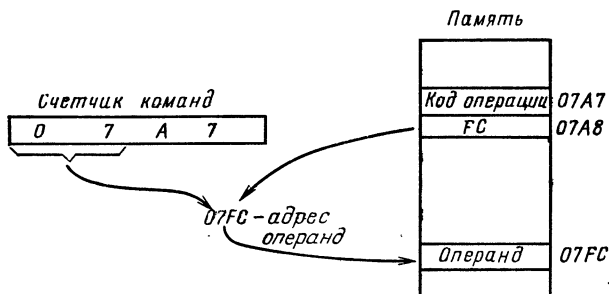


Рис. 13.7.

зать значение второго байта адреса памяти. Пример, приведенный на рис. 13.7, иллюстрирует случай, когда на странице содержится 256 слов. Таким образом, максимальным значением второго байта адреса операнда будет FF_{16} . Счетчик команд содержит адрес $07A7_{16}$; это адрес кода операции. За кодом операции следует значение второго

байта адреса операнда FC_{16} . Значение первого байта адреса операнда задается первым байтом содержимого счетчика команд (07_{16}). Таким образом, полный адрес операнда $07FC_{16}$.

П р и м е ч а н и е. Этим методом можно определить только адреса операндов, находящиеся на той же странице, что и код операции.

Во всех случаях, когда требуется адресоваться за пределы данной страницы, необходимо использовать другой метод адресации, например метод прямой адресации.

Индексная адресация. При индексной адресации действительный адрес операнда находится прибавлением числа, следующего за кодом операции, к числу в *индексном ре-*

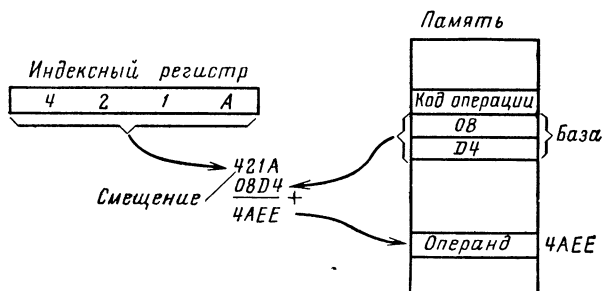


Рис. 13.8.

гистре (рис. 13.8). Адрес 0804_{16} , заданный в команде, прибавляется к адресу $421A_{16}$ из индексного регистра, и с действительным адресом операнда будет $4AEE_{16}$. Индексная адресация применяется в том случае, когда необходимо записать или считать список данных из последовательных ячеек памяти. Можно адресовать каждую ячейку памяти, используя прямую адресацию, однако в этом случае каждая команда будет 3-байтной. Более того, придется включать в программу, а также в память дополнительные команды каждый раз, когда производится обращение к памяти. Применение индексной адресации требует меньше времени (при подготовке программы) и занимает меньше места в памяти. Сделаем базовый адрес равным адресу первого числа списка данных и установим индексный регистр в 0. Теперь можно ввести команду, которая реализует считывание из памяти методом индексной адресации. При этом считывается первое число. Затем содержимое индексного регистра

инкрементируется и повторяется предыдущая команда. Считывается второе число списка данных. Эти команды могут повторяться до тех пор, пока не будет считан весь список.

Выводы

1. Относительная адресация — это сводное название ряда методов, имеющих общий принцип, заключающийся в том, что действительный адрес операнда в памяти определяется суммированием базового адреса и смещения, заданного в команде.

2. Двумя типами относительной адресации являются постраничный и индексный методы адресации.

3. При постраничной адресации память разделяется на блоки, содержащие определенное число машинных слов.

Когда код команды и операнд команды находятся на одной странице, адрес операнда определяется по первому байту счетчика команд и байту, следующему за кодом операции.

4. При индексной адресации действительный адрес операнда находится путем сложения содержимого индексного регистра с базовым адресом, следующим за кодом операции.

Глава четырнадцатая БЛОК-СХЕМА АЛГОРИТМА

14.1. Введение

Когда требуется выполнить на ЭВМ некоторую работу, начинают с постановки задачи. Прежде чем ЭВМ начнет выполнять вычисления, связанные с решением задачи, задача должна быть детально проработана. Уровень проработки задачи должен быть таким, чтобы на его основе можно было легко написать программу.

После тщательной проработки задачи выбранный для ее решения метод может быть изображен в виде блок-схемы. В данной главе описывается процесс построения блок-схемы.

14.2. Символы, используемые в блок-схеме

Блок-схема представляет собой упорядоченную совокупность блоков, соединенных между собой. В каждом

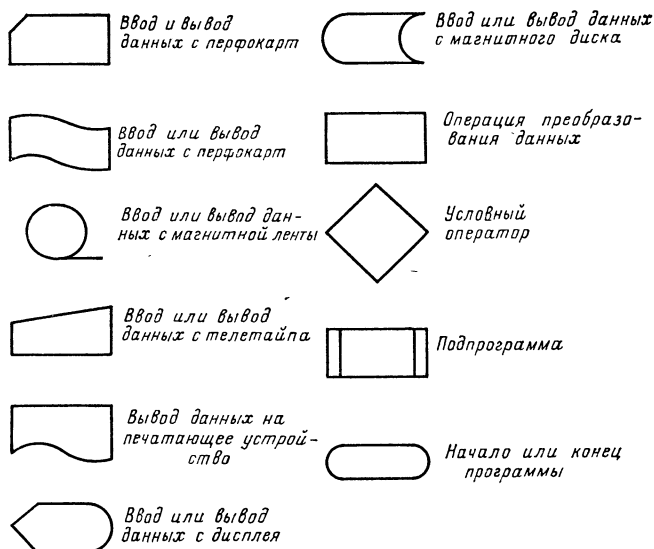


Рис. 14.1.

блоке содержится описание одной или нескольких операций. Примером выполняемой операции могут служить сложение двух чисел, проверка выполнения некоторого условия и т. п.

Придавая различным блокам различную форму, даем зрительное представление о том, какую операцию выполняет тот или иной блок. На рис. 14.1 приведены стандартные обозначения блоков, используемых при построении блок-схем, и даны объяснения выполняемых ими действий.

Стрелка на блок-схеме показывает, в каком направлении развиваются события в программе. На рис. 14.2 приведен пример типичной блок-схемы для циклической программы. При каждом выполнении цикла сначала выполняется

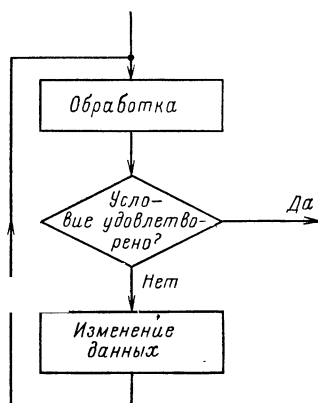


Рис. 14.2.

требуемая операция. Затем проверяется, выполнено ли заданное условие. Если условие не выполнено, то происходит изменение данных (например, прибавляется 1). Затем операция повторяется и анализируется вновь полученный результат. Процесс продолжается до тех пор, пока не будет выполнено заданное условие, после чего программа продолжает свою работу, переходя по стрелке, помеченной словом *Да*.

14.3. Типы блок-схем

На этапе постановки задачи необходимо построить блок-схему, которая с точностью до отдельных выполняемых действий соответствует будущей программе. Другими сло-

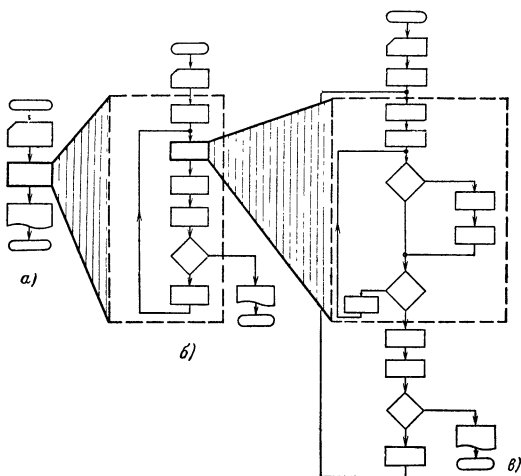


Рис. 14.3.

вами, каждому блоку в блок-схеме должна соответствовать одна или несколько команд из системы команд машины. Поскольку практически невозможно сразу построить такую блок-схему, следует начать с укрупненной блок-схемы, отражающей общую постановку задачи. В этом случае каждый блок соответствует некоторой совокупности процессов или операций. Затем детализируют (подразделяют) каждый блок, пока не будет достигнуто соответствие одного блока одной команде или группе команд в программе.

Естественно, что блок-схема программы, написанной на языке высокого уровня, будет менее подробной, чем блок-схема программы, написанной на языке ассемблера, поскольку один оператор языка высокого уровня соответствует нескольким машинным командам.

Можно выделить три типа блок-схем (рис. 14.3).

1. Системные блок-схемы.

В системной блок-схеме показывается, какие устройства используются для ввода, вывода и хранения данных. Алгоритм решения задачи изображается в виде единственного блока (рис. 14.3 а).

2. Укрупненные (основные) блок-схемы.

В таких блок-схемах алгоритм проработан на принципиальном уровне (рис. 14.3, б).

3. Подробные (детальные) блок-схемы.

В блок-схемах этого типа происходит дальнейшая детализация алгоритма, описанного укрупненной блок-схемой, так что каждый блок представляет одну или иногда две-три команды (рис. 14.3, в).

Выводы

1. Метод решения задачи может быть изображен в виде блок-схемы.

2. Блок-схема представляет собой упорядоченную совокупность блоков, соединенных между собой. Стрелки показывают направление развития вычислительного процесса.

3. Блок-схемы бывают трех типов: системные, укрупненные (основные), подробные (детальные).

4. Действие, указанное в каждом блоке подробной блок-схемы, может быть выполнено с помощью одной (или нескольких) команд из системы команд машины.

14.4. Задача

Рассмотрим следующую задачу. Имеется пять перфокарт, расположенных в случайном порядке. На каждой перфокарте имеется число, принадлежащее интервалу от 1 до 99. Электронная вычислительная машина должна упорядочить эти перфокарты по возрастанию значений нанесенных на них чисел и напечатать результат.

14.5. Задача сортировки

Чтобы сконструировать блок-схему задачи, необходимо представить себе, как ЭВМ ее решает. Перед тем как приступить к разработке блок-схемы программы сортировки, опишем сначала в принципе, как ЭВМ решает задачу последовательного упорядочения чисел. Для простоты рассмотрим это на примере упорядочения трех чисел.

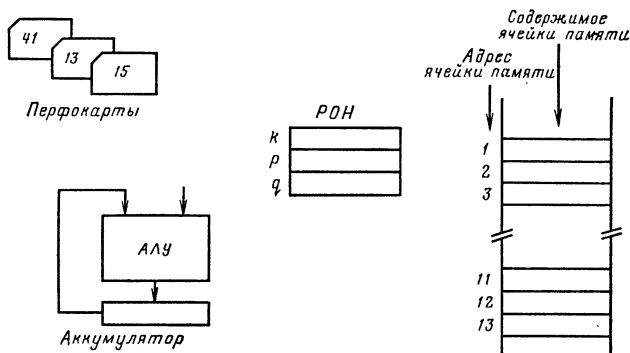


Рис. 14.4.

Решение задачи иногда упрощается, если ее представить графически. Так и поступим с задачей сортировки, используя при этом терминологию и функции, присущие ЭВМ (рис. 14.4).

Числа в случайном порядке вводятся в последовательные ячейки памяти в секции Р. Первое вводимое число записывается по адресу 1. Второе и третье числа записываются соответственно по адресам 2 и 3 (рис. 14.5).

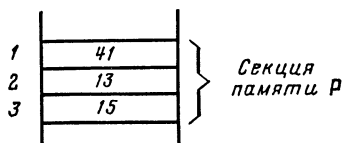


Рис. 14.5.

Электронная вычислительная машина с помощью одной операции может сравнить два числа. При выполнении операции сравнения число из аккумулятора сравнивается с числом, выбираемым из памяти. Операция сравнения выполняется в АЛУ.

Чтобы ЭВМ могла многократно выполнять одну и ту же операцию (например, сравнивать числа из памяти с числом, находящимся в аккумуляторе), в аккумулятор перед

началом сравнения следует загрузить соответствующее значение. Поскольку необходимо упорядочить перфокарты по возрастанию, следует загрузить в аккумулятор число, которое заведомо больше любого из упорядочиваемых чисел. Поэтому загрузим в аккумулятор число 100. Затем сравним содержимое ячейки памяти, находящейся по адресу 1, с содержимым аккумулятора. Меньшее из двух чисел поместим в аккумулятор: это будет число 41. Далее будем сравнивать содержимое ячейки памяти по адресу 2 с содержимым аккумулятора. Меньшее из двух чисел (41 и 13) зашлем в аккумулятор: в аккумуляторе будет находиться число 13. Наконец, сравним содержимое ячейки памяти, имеющей адрес 3, с содержимым аккумулятора, и опять зашлем в аккумулятор меньшее из сравниваемых чисел (15 и 13). В результате в аккумуляторе будет находиться число 13.

Как только эта операция будет выполнена, то, поскольку числа должны быть упорядочены по возрастанию, перешлем содержимое аккумулятора (число 13) в первую ячейку памяти секции Q.

Следующее наименьшее число определяется в процессе второго прохода сортировки. Это можно успешно сделать только в том случае, если из секции Р было удалено ранее найденное минимальное число, так как в противном случае оно вновь будет участвовать в операции. Чтобы этого не произошло, загрузим в ту ячейку памяти, где находилось минимальное число, число, которое заведомо больше любого из сортируемых чисел. В данном случае это число 100. Когда будет найдено второе наименьшее число, то на его место также запишем число 100 и т. д. При этом предполагается, что ЭВМ помнит адрес ячейки памяти, в которой было расположено последнее число. Электронно-вычислительная машина обеспечивает это, засылая адрес числа, хранимого в аккумуляторе, во вспомогательную регистровую пару и, корректируя ее содержимое всякий раз, когда содержимое аккумулятора меняется, г. е. в тот момент, когда обнаруживается меньшее число. Таким образом, во вспомогательной регистровой паре всегда содержится адрес наименьшего числа, найденного в процессе сортировки.

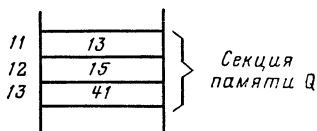


Рис. 14.6.

Число операций сравнения при поиске минимального числа всегда равно числу введенных карт, в данном случае — 3. В результате второго прохода сортировки найденное значение помещается по адресу 12.

После завершения процесса сортировки числа 41, 13 и 15 расположатся в памяти в секции Q в порядке возрастания (рис. 14.6).

Проделанный анализ алгоритма сортировки позволяет сделать следующее заключение:

1) перед началом сортировки следует загрузить в аккумулятор число, которое по меньшей мере на 1 больше самого большого из сортируемых чисел;

2) число операций в одном проходе сортировки равно числу сортируемых элементов;

3) число проходов сортировки также равно числу сортируемых элементов;

4) найденное во время очередного прохода сортировки число следует заменить в секции P памяти числом, которое по меньшей мере на 1 больше самого большого из сортируемых чисел;

5) следует определить, по какому адресу необходимо засылать первое введенное число, а также по какому адресу следует засылать число, найденное в первом проходе сортировки.

Выводы

1. Электронно-вычислительная машина может в одной операции сравнить только два числа.

2. Сравнение двух чисел происходит в АЛУ. При этом одно число должно быть загружено в аккумулятор, а другое выбирается из памяти.

3. В рассмотренном методе сортировки число проходов сортировки равно числу сортируемых элементов. Число выполняемых в каждом проходе сравнений также равно числу элементов сортируемой последовательности.

14.6. Системная блок-схема

Системная блок-схема для данной задачи показана на рис. 14.7. Главная цель этой схемы состоит в том, чтобы показать, какие устройства используются для ввода и вывода данных.

Вводимые данные представляют собой совокупность из пяти чисел, нанесенных на перфокарты и принимающих

значения в интервале 1—99. Данные вводятся с помощью устройства ввода с перфокарт и помещаются в память ЭВМ. Алгоритм сортировки представлен на этой блок-схеме блоком «сортировки чисел». Результат сортировки выводится с помощью печатающего устройства.

14.7. Основная блок-схема

Основная блок-схема алгоритма решения задачи показана на рис. 14.8.

Получив указание начать работу (блок 1), ЭВМ выполняет команду чтения чисел, набитых на перфокартах, и размещает их в секции Р памяти (блок 2).

Прежде чем начать процесс сортировки, следует указать, по какому адресу будет послано наименьшее число, найденное в первом проходе сортировки. Укажем ад-

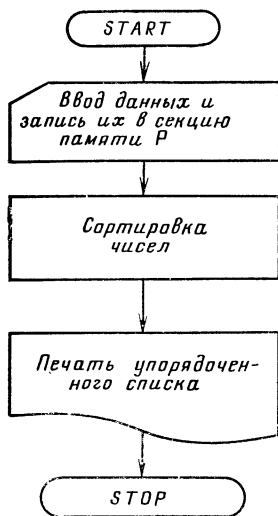


Рис. 14.7.

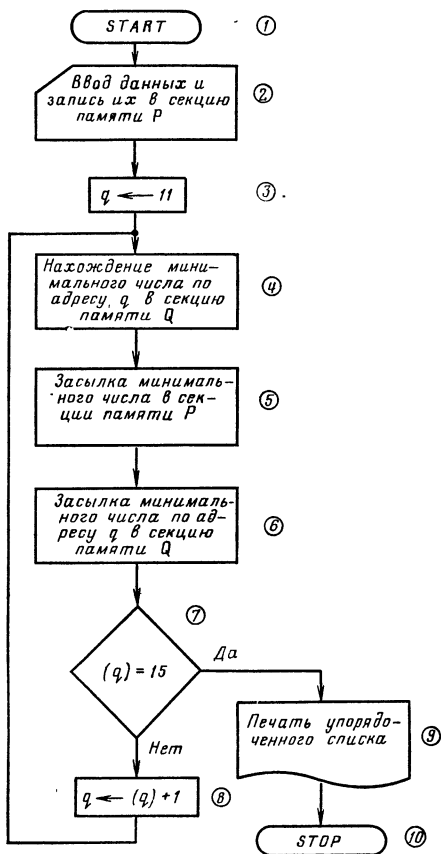


Рис. 14.8.

рес 11. Слово, расположенное по этому адресу, находится в секции памяти Q. Для запоминания адреса слова, находящегося в секции Q, можно использовать пару регистров блока РОН. Обозначим эту пару регистров q . Содержимое этой пары РОН обозначим (q). Таким образом, (q) есть тот адрес, по которому следует поместить самое маленькое число, найденное при выполнении прохода сортировки.

П р и м е ч а н и е. В рассматриваемом примере первое слово в секции Q имеет адрес 11. С таким же успехом можно было бы выбрать адрес $03FF_{16}$. Тогда блок 3 содержал бы оператор $q \leftarrow 03FF_{16}$. Блок 4 выполняет операцию поиска наименьшего числа в секции памяти P. В блоке 5 осуществляется пересылка этого числа в секцию памяти Q по адресу, который хранится в паре РОН q . В блоке 6 минимальное число, найденное в секции памяти P, заменяется на число 100, чтобы не произошло его выборки во время очередного прохода операции сортировки.

Содержимое регистровой пары q , в которой записан адрес элемента памяти из секции Q, увеличивается на 1 при каждом выполнении цикла. Цикл образован блоками 4—8 (включительно). Регистровая пара q все время содержит адрес, по которому следует заслать очередное число.

После того, как были выполнены действия, заданные в блоках 4—6, результат сравнения в блоке 7 (сравнивается содержимое пары РОН q с числом 15) указывает, отсортированы или нет все пять чисел. Наименьшее из пяти чисел расположено по адресу 11, а самое большое по адресу 15.

До первого выполнения цикла содержимое регистровой пары равно 11. Оно увеличивается на 1 после каждого прохода цикла. После четвертого прохода оно равно 15. После четырехкратного прохода цикла программа выходит из блока 7 в направлении Да. Затем печатается отсортированный список (блок 8).

Выводы

1. Системная блок-схема предназначена для того, чтобы показать какие устройства ввода-вывода требуются при решении задачи. Задача описывается одним блоком.

2. Укрупненная блок-схема описывает метод решения задачи в общих чертах.

14.8. Детальная блок-схема

При разработке подробной блок-схемы происходит дальнейшее расчленение тех блоков укрупненной блок-схемы, которые не могут быть выполнены с помощью одной или двух команд. Уровень детализации определяется системой команд микро-ЭВМ.

Подробная блок-схема задачи, рассматриваемой в данной главе, показана на рис. 14.9. Рассмотрим эту блок-схему последовательно.

Сравнение двух чисел, одно из которых находится в аккумуляторе, выполняется в АЛУ. В блок-схеме аккумулятор будем обозначать символическим именем АСС.

Блоки 4—6 (см. рис. 14.8) должны быть детализированы. Операции ввода и вывода (блоки 2 и 9) также следовало бы раскрыть более подробно. Однако, поскольку ЭВМ имеет для этой цели специальные программы (например, хранимые в ПЗУ), оставим эти блоки неизменными.

Блок 1 (рис. 14.9) указывает начальную точку программы.

Блок 2. Сортируемые числа вводятся и записываются в секцию памяти Р по адресам 1—5.

Блок 3 может быть без изменений взят из укрупненной блок-схемы, так как его выполнение обеспечивается одной командой. В этом блоке выполняется загрузка в пару регистров q числа 11. В этой паре регистров содержится адрес размещения очередного минимального числа. Самое маленькое число, засылаемое после первого прохода сортировки, будет послано по адресу 11.

Блок 4. Перед началом выполнения прохода сортировки в аккумулятор следует послать число большее, чем самое большое из сортируемых чисел. В рассматриваемом примере следует загрузить в аккумулятор число 100.

Блок 5. Введенные числа были размещены в секции памяти Р по адресам 1—5. Адрес слова из секции Р, которое будет сравниваться с содержимым аккумулятора, загружается в пару регистров p . Содержимое регистра p обозначается (p).

В блоке 5 в пару регистров p загружается число 1, так как сначала содержимое аккумулятора должно сравниваться с первым числом.

При первом сравнении, выполняемом в АЛУ, из памяти выбирается число по адресу 1, при втором сравнении по адресу 2 и т. д. Заметим, что число в паре регистров p

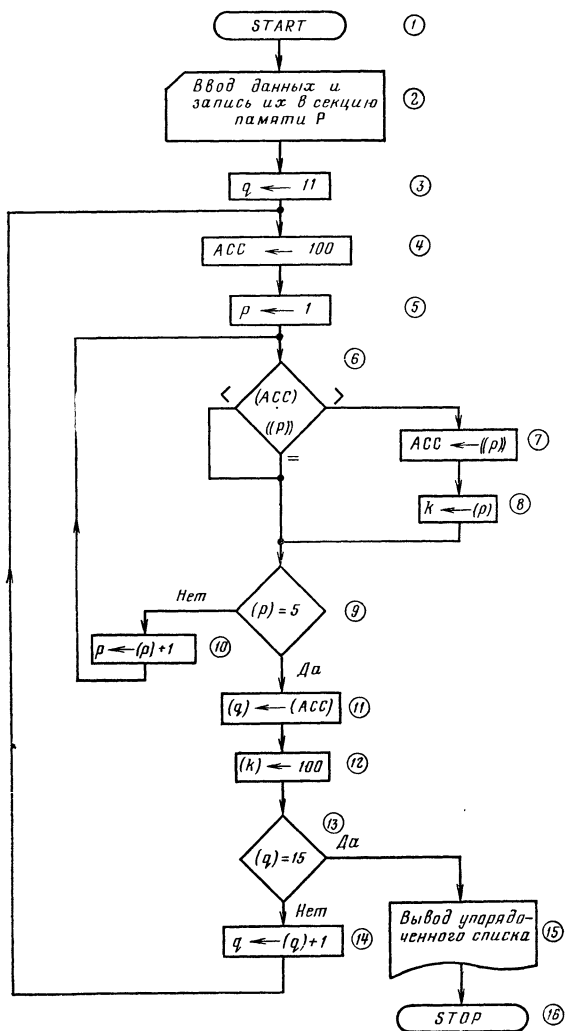


Рис. 14.9.

представляет собой также число выполненных сравнений. Так как содержимое p после каждого сравнения увеличивается на 1, то, сравнивая его с числом 5, обнаружим тот момент, когда было выполнено пять сравнений.

Примечание. Если бы, например, числа занесли в память по адресам 0201₁₆—0206₁₆, а не по адресам 1—5, то в блоке 5 следовало бы загрузить в пару регистров p число 0201₁₆. Для проверки выполнения пяти сравнений следовало бы сравнивать содержимое пары регистров p с числом 0205₁₆.

Блоки 6—10. В блоке 6 содержимое аккумулятора сравнивается с содержимым ячейки памяти, адрес которой находится в паре регистров p . Скобки подчеркивают, что операндами являются содержимое аккумулятора и содержимое ячейки памяти.

Перед первым выполнением программного цикла, образованного блоками 6, 9 и 10 (см. рис. 14.9), содержимое пары регистров p равно 1 и показывает, по какому адресу расположено первое из сортируемых чисел.

Перед первым выполнением операции сравнения в аккумуляторе загружается число 100. После первого выполнения операции, указанной в блоке 6, результат сравнения всегда будет свидетельствовать о том, что содержимое аккумулятора больше числа, с которым производилось сравнение.

В блоке 7 осуществляется пересылка содержимого ячейки памяти, которая участвовала в операции сравнения в блоке 6, в аккумулятор. В результате содержимое аккумулятора меняется. Адрес засылаемого числа помещается в пару регистров k . После выполнения операции сравнения содержимое этой пары регистров представляет собой адрес ячейки памяти в секции Р, в которой хранится меньшее из двух сравниваемых чисел. Если после выполнения следующей операции сравнения будет найдено еще меньшее число (т. е. при выходе из блока 6 программа продолжается в направлении, помеченном символом «>»), то содержимое пары регистров k будет, конечно, изменено на адрес этого меньшего числа.

В начале программы была выделена пара регистров для хранения адресов памяти секции Р, в которой находятся сортируемые числа. При первом выполнении сравнения содержимое этой пары регистров равно 1, так как первое число, которое сравнивается с содержимым аккумулятора, находится по адресу 1 в секции памяти Р. После каждого

прохода цикла содержимое этой пары регистров увеличивается на 1 и, следовательно, в ней всегда находится адрес следующего сравниваемого числа. Увеличение на 1 происходит в блоке 10. Программный цикл, образованный блоками 6, 9 и 10, начинается в точке, соответствующей выходу из блока 9, помеченному словом *Нет*. После первого прохода цикла содержимое пары регистров p будет равно 2. После четырех проходов цикла содержимое этого регистра станет равным 5 и при выходе из блока 9 программа будет продолжаться в направлении, помеченном словом *Да*. В результате выполнено пять операций сравнения.

Блок 11. Здесь минимальное число должно быть помещено в секцию памяти Q по адресу 11. Блок 11 выполняет пересылку содержимого аккумулятора по адресу 11.

Блок 12. Если вновь повторим операцию поиска минимального числа, то найдем то же самое минимальное число из секции памяти R. Следовательно, необходимо исключить это число, засылая в соответствующую ячейку памяти число, которое заведомо больше любого из оставшихся чисел. В блоке 12 по адресу, указанному в паре регистров k , посылается число 100. По условиям задачи это число никогда не сможет оказаться минимальным.

Блоки 13 и 14. В блоке 13 с помощью операции сравнения проверяется, стало ли равным 15 содержимое регистровой пары q . Другими словами, проверяется, сделано ли пять проходов сортировки. Если это не так, то содержимое q увеличивается на 1 и начинается следующий проход сортировки. Если содержимое q равно 15 (выполнено пять проходов сортировки), то в адресах памяти 11—15 в секции находятся упорядоченные по возрастанию числа.

Блоки 15 и 16. Если $(q) = 15$, то результат сортировки последовательно выводится на печать. Блок 16 останавливает процесс.

Выводы

1. Каждому блоку подробной блок-схемы соответствует одна (в отдельных случаях две или три) команды из системы команд машины.

2. Блоки ввода-вывода (состоящие из нескольких команд) обрабатываются специальными встроенными программами микро-ЭВМ.

3. Блок-схема всегда начинается с блока START («Начало») и заканчивается блоком STOP («Конец»).

Глава пятнадцатая

ОТ ПОСТАНОВКИ ЗАДАЧИ К РЕШЕНИЮ

15.1. Введение

Если задача поставлена, то для ее решения необходимо в заданном порядке пройти через несколько этапов разработки, прежде чем будет получена исходная программа. Это следующие этапы:

1. Описание задачи.
2. Анализ задачи.
3. Составление системной блок-схемы алгоритма.
4. Составление основной блок-схемы алгоритма.
5. Составление детальной блок-схемы алгоритма.
6. Превращение детальной блок-схемы в исходную программу.

В этой главе на примере покажем действия разработчика на всех этапах разработки программы.

15.2. Описание задачи

В качестве примера используем систему измерения времени на стадионе. На беговую дорожку выходит спортсмен, который собирается побить мировой рекорд в беге на 100 м

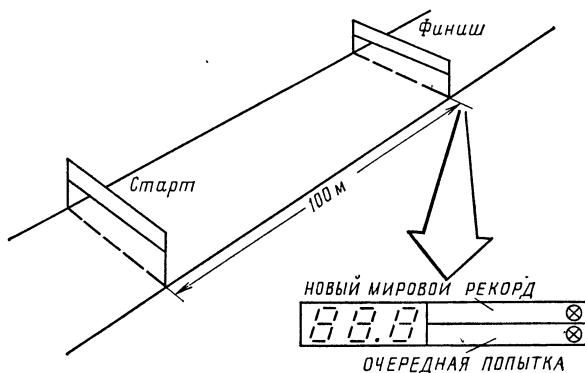


Рис. 15.1.

(рис. 15.1). Задача состоит в том, чтобы с помощью микро-ЭВМ определить его время и зафиксировать факт установления нового мирового рекорда. Измерение времени должно быть выполнено с точностью до десятых долей секунды.

15.3. Анализ задачи

Микро-ЭВМ должна выполнить следующие действия:

а) определить время, затраченное бегуном на путь от старта до финиша, и отобразить его на световом табло;

б) сравнить это время с прежним мировым рекордом. Отобразить на табло результат этого сравнения путем зажигания транспонантов **НОВЫЙ МИРОВОЙ РЕКОРД** или **ОЧЕРЕДНАЯ ПОПЫТКА**.

Входная информация для микро-ЭВМ:

а) имеется аппаратура, генерирующая сигнал при пересечении бегуном стартовой линии;

б) имеется аппаратура, генерирующая сигнал при пересечении бегуном линии финиша.

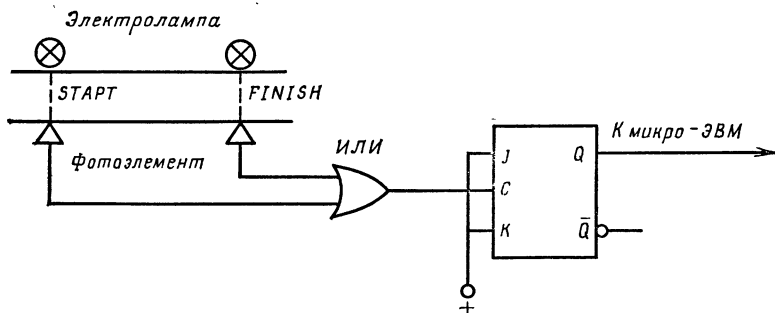


Рис. 15.2.

Эта аппаратура может представлять собой, например, комбинацию из электроламп и фотозлементов. К-триггер и схема ИЛИ используются для того, чтобы ввести эти сигналы в микро-ЭВМ (рис. 15.2).

На выходе Q триггера будет 1 при пересечении бегуном стартовой линии и 0 при пересечении бегуном линии финиша. Таким образом, время, в течение которого на выходе Q триггера будет 1, и будет временем, которое затратит бегун на дистанцию 100 м. Перед стартом триггер сбрасывается и на его выходе появляется 0. В этом примере вычисляемое время есть время, в течение которого триггер находится в состоянии 1.

Выходная информация может быть представлена визуально с помощью нескольких 7-сегментных элементов линейного дисплея (см. рис. 15.1).

Выводы

1. Чтобы получить исходную программу, необходимо прежде всего описать и проанализировать задачу, разработать блок-схему алгоритма и преобразовать ее в исходную программу.

2. Детализация способов формирования входной и выходной информации также выполняется на этапе анализа задачи.

15.4. Составление блок-схемы

Блок-схема алгоритма представляет метод, выбранный для решения задачи. Довольно часто бывает, что одна и та же задача может быть решена различными методами. Способ решения задачи, выбранный на этапе ее анализа, на основе которого строится блок-схема алгоритма, определяет качество разрабатываемой программы.

Существуют три вида блок-схем алгоритмов:

а) системная блок-схема, в которой специфицируются устройства, используемые для ввода и вывода информации. На этой блок-схеме собственно задача представляется одним блоком;

б) основная блок-схема, в которой задача описывается более подробно;

в) детальная блок-схема, в которой каждый блок содержит одну или несколько команд из набора команд используемой ЭВМ.

Системная блок-схема представлена на рис. 15.3.

Расчленим блок «Вычисление времени» на небольшие блоки таким образом, чтобы выбранный метод решения задачи был четко представлен на блок-схеме. Основная блок-схема алгоритма показана на рис. 15.4, а, детальная блок-схема — на рис. 15.4, б.

Содержимое регистра A увеличивается на 1 каждую 0,1 с до тех пор, пока на выходе Q не появится 0, другими сло-

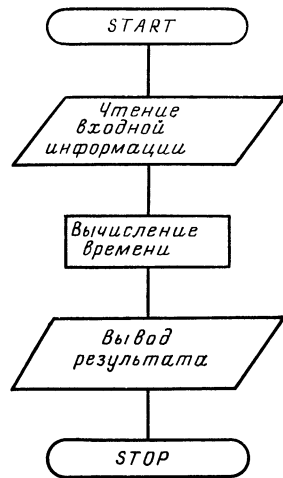


Рис. 15.3.

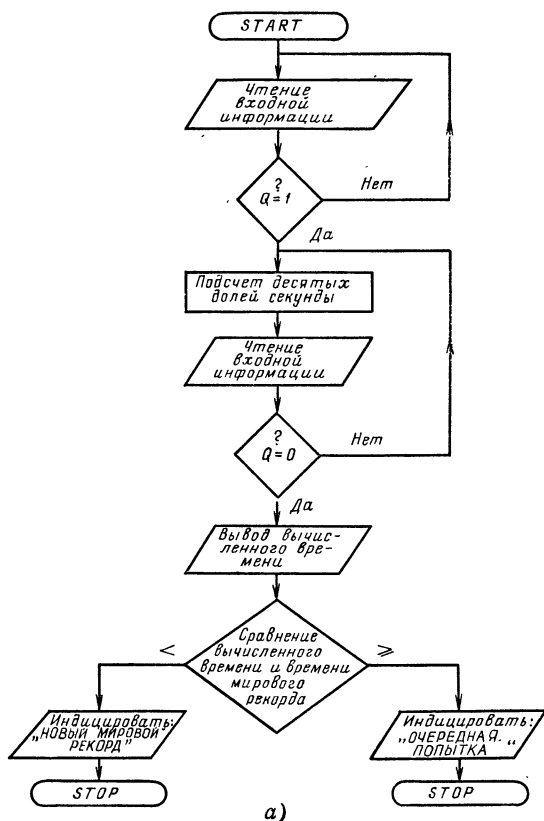


Рис. 15.4.

вами, пока бегун не пересечет линию финиша. Число X имеет размерность 0,1 с. При расчете величины X используется метод, описанный в гл. 16.

15.5. Написание исходной программы

Рассмотрим этап разработки, на котором необходимо перейти от описания решения задачи, представленного в виде детальной блок-схемы алгоритма, к языку, который «понимает» ЭВМ. Однако это не означает, что программист должен написать последовательности 0 и 1. Недостатки написания программы на языке объектных кодов рассматри-

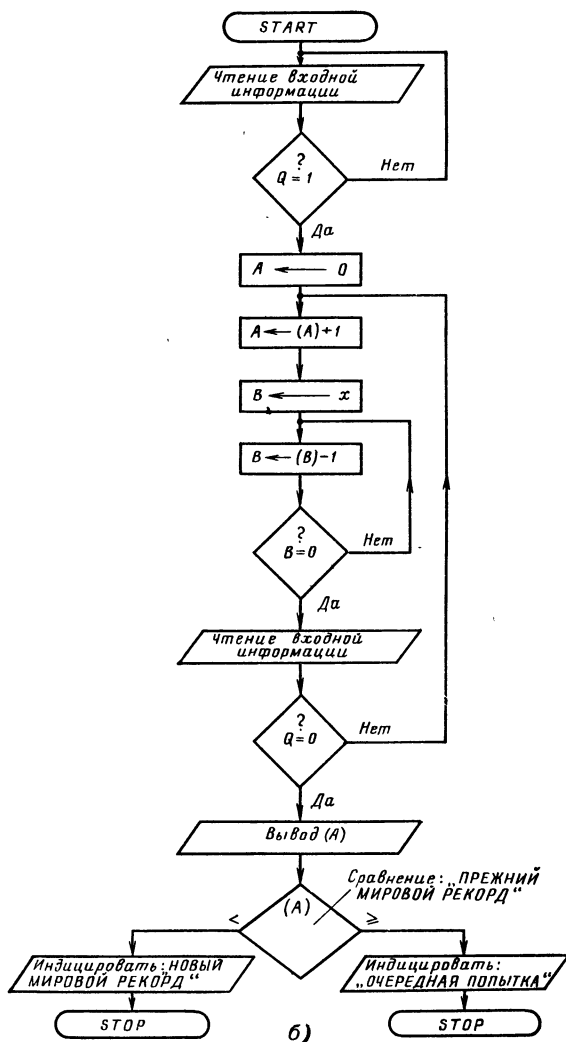


Рис. 15.4.

вались в предыдущих главах. Программист может перевести блок-схему на один из языков программирования, а программа-транслятор преобразует исходную программу в объектную.

Наиболее часто используемый в микро-ЭВМ язык программирования — это язык ассемблера. Программа-тран-

Метка	Мнемо-код	Операнд	Комментарий
	ORG	8200H	; Первая команда с адресом 8200H
WR10:	EQU	00H	; Прежний мировой рекорд равен 9,9 с
WRO, 1:	EQU	98H	
	LXI	SP, 8400H	; Определение стека
BEGIN	LXI	B, 0000H	; Сброс счетчика
START:	IN	01H	; Читать Q
	ANI	01H	
	JZ	START	; Переход, если Q=0
TEL:	CALL	WATCH	; Ждущий цикл 0,1 с
	MOV	A, C	
	ADI	01H	; Содержимое C + 0,1 с
	DAA		; Десятичная коррекция
	MOV	C, A	
	MOV	A, B	
	ACI	00H	; Содержимое B + перенос (увеличение десятков на 1)
	DAA		; Десятичная коррекция
	MOV	B, A	
	IN	01H	; Читать Q
	ANI	01H	
	JNZ	TEL	; Переход, если Q ≠ 0
	MOV	A, B	
	OUT	02H	; Вывод десятков
	MOV	A, C	
	OUT	03H	; Вывод единиц и десятых долей
	MVI	D, WR10	; Прежний мировой рекорд в регистровой паре D, E
	MVI	E, WRO, 1	
	MOV	A, B	
	CMP	D	; Сравнить десятки
	JNC	AGAIN	; Переход, если (B) > (D)
	MOV	A, C	
	CMP	E	; Сравнить единицы и десятые доли
	JNC	ACAIN	; Переход, если (C) > (E)
	MVI	A, 01H	
	OUT	04H	; Индексировать НОВЫЙ МИРОВОЙ РЕКОРД
AGAIN:	JMP	BEGIN	
	MVI	A, 02H	
	OUT	04H	; Индексировать ОЧЕРЕДНАЯ ПОПЫТКА
	JMP	BEGIN	
WATCH:	MVI	D, 32H	
LUS2:	MVI	E, FFH	; 32FFH в регистровой паре D, E
LUS1:	DCR	E	; Уменьшить на 1 содержимое регистра E
	JNZ	LUSI	
	DCR	D	; Уменьшить на 1 содержимое регистра D
	JNZ	LUS2	
	RET		
	END		; Конец программы

слятор, которая переводит исходную программу в объектную, и называется ассемблером или ассемблирующей программой. Далее такую программу будем называть ассемблером.

Прежде чем приступить к написанию программы, необходимо специфицировать память, т. е. определить:

а) адрес ячейки памяти, в которую помещается первая команда программы;

б) какую часть памяти нужно отвести под стек, если его наличие необходимо;

в) какие РОН или ячейки памяти нужно использовать для хранения переменных.

Когда это будет сделано, можно взять детальную блок-схему алгоритма, список команд данной микро-ЭВМ и (наконец-то!) приступить к программированию. Результат этого этапа работы (исходная программа на языке ассемблера) приведен в табл. 15.1.

15.6. От исходной программы к решению

Теперь имеется исходная программа на бумаге. Чтобы микро-ЭВМ имела возможность выполнить ее, необходимо проделать следующие операции:

1. Перенести исходную программу на перфоленту в символах кода ASCII. Это может быть сделано с помощью программы-редактора.

2. Транслировать исходную программу в объектную с помощью ассемблера.

3. Тестировать программу с помощью программы-отладчика.

Эти операции рассмотрены в гл. 18.

Выводы

1. Сначала составляется системная блок-схема алгоритма, затем основная и, наконец, детальная.

2. Исходная программа получается последовательным переводом каждого блока детальной блок-схемы на язык ассемблера.

3. Исходная программа на бумаге должна быть последовательно перенесена на перфоленту, транслирована в объектную программу и затем отлажена.

Глава шестнадцатая

ПРИМЕРЫ ПРОГРАММ

Чтобы читатель чувствовал себя увереннее при составлении блок-схем и при написании программ, разберем пять примеров программ, с которыми часто придется сталкиваться на практике.

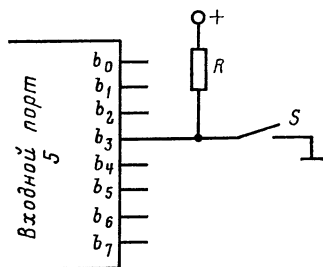


Рис. 16.1.

Предположим, что читатель уже ознакомился с материалом гл. 11.

Пример 16.1.

Дано. Тумблер присоединен ко входу b_3 порта ввода 5 (рис. 16.1). Если тумблер разомкнут, то $b_3 = 1$; если он замкнут, то $b_3 = 0$.

Требуется. Написать программу, которая про-

верит значения сигнала на входе b_3 порта ввода 5 и осуществит переход по программе к части А в том случае, если $b_3 = 0$, и к части В, если $b_3 = 1$.

Решение. Блок-схема решения этой задачи показана на рис. 16.2. Назовем эту программу TESTSW (тест ключа) и используем это символическое имя в качестве метки, указывающей на адрес первой команды программы. Если это необходимо, то можно обращаться к этой программе в процессе выполнения другой программы, используя команду CALL TESTSW.

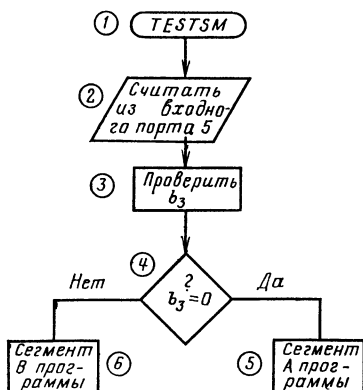


Рис. 16.2.

В блоке 2 блок-схемы алгоритма (рис. 16.2) информация из порта ввода 5 передается в аккумулятор, где состояние входа b_3 может быть проверено маскированием всех остальных бит входного слова (блок 3). Если $b_3 = 0$, то программа продолжается в направлении *Да* от блока 4 и переходит

Метка	Мнемокод	Операнд	Комментарий
TESTSW:	IN	05H	; Ввод из порта 5 в аккумулятор
	ANI	08H	; Маскирование $b_7, b_6, b_5, b_4, b_2, b_1$ и b_0
	JZ	A	; Переход к сегменту A, если $b_3=0$, иначе переход к следующей команде
B:	—		; Первая команда сегмента B программы
	—		
A:	—		; Первая команда сегмента A программы
	—		

к секции A программы (блок 5). Если $b_3 = 1$, то программа продолжается в направлении *Нет* и переходит к секции B программы (блок 6). Листинг программы приведен в табл. 16.1.

Пример 16.2. Д а н о. Схема, показанная на рис. 16.3.

Т р е б у е т с я. Написать подпрограмму, которая постоянно опрашивает состояние b_5 до

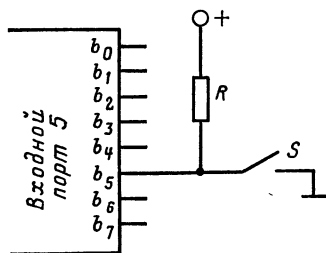


Рис. 16.3.

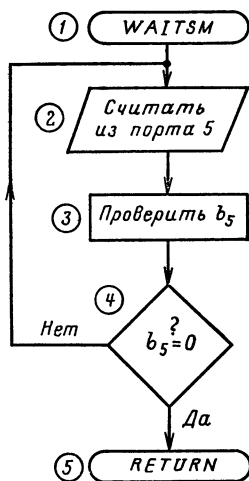


Рис. 16.4.

тех пор, пока оно не станет равным нулю, т. е. пока тумблер не будет замкнут (рис. 16.3). Осуществить возврат в основную программу, если тумблер замкнут ($b_5 = 0$).

Р е ш е н и е. Блок-схема решения этой задачи показана на рис. 16.4. Назовем эту подпрограмму WAITSW, т. е. wait switch (ожидание включения), и используем это

Метка	Мнемокод	Операнд	Комментарий
WAITSW:	IN	05H	; Ввод из порта 5 в аккумулятор
	ANI	20H	; Маскирование $b_7, b_6, b_4, b_3, b_2, b_1$ и b_0
	JNZ	WAITSW	; Переход к циклу ожидания, если $b_5 = 1$
	RET		; Возврат к основной программе, если $b_5 = 0$

символическое имя в качестве метки, указывающей на первую команду этой подпрограммы.

В блоке 2 байт, присутствующий на входе порта ввода 5, считывается и засылается в аккумулятор. Состояние входа b_5 анализируется в аккумуляторе маскированием. Если $b_5 = 1$, то подпрограмма из блока 4 продолжается в на-

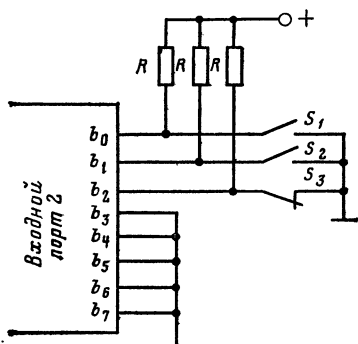


Рис. 16.5.

правлении *Нет* и операция проверки повторяется до тех пор, пока b_5 не станет равным нулю. В этом случае подпрограмма продолжится в направлении *Да* от блока 4 и будет произведен возврат к основной программе с помощью команды RETURN.

Эта программа показана в табл. 16.2.

Пример 16.3. Дан о. Три тумблера подсоединены ко входам b_0, b_1 и b_2 входного порта 2 (рис. 16.5).

Если тумблер включен, то на соответствующем входе 0. Если тумблер не включен, то на входе 1. Входы $b_3 - b_7$ заземлены, т. е. на них постоянно подается 0.

Таким образом, с помощью трех тумблеров имеется возможность представить восемь различных состояний процесса:

$00000000_2 = 00_{16}$
 $00000001_2 = 01_{16}$
 $00000010_2 = 02_{16}$
 $00000011_2 = 03_{16}$
 $00000100_2 = 04_{16}$
 $00000101_2 = 05_{16}$
 $00000110_2 = 06_{16}$
 $00000111_2 = 07_{16}$

Т р е б у е т с я . Написать программу перехода к одной из восьми подпрограмм (от А до Н) в зависимости от комбинации сигналов во входном порте 2.

Р е ш е н и е . Предположим, что соответствующие подпрограммы расположены в памяти так, как это показано на рис. 16.6. Первая команда подпрограммы А находится в ячейке с адресом 0300_{16} ; первая команда подпрограммы В находится в ячейке с адресом 0400_{16} и т. д. Адреса первых команд подпрограмм хранятся в другой части памяти так, как это показано на рис. 16.7. Для записи каждого 16-разрядного адреса используются две ячейки памяти. Адрес первой команды подпрограммы А хранится в ячейках памяти с адресами 0202_{16} и 0201_{16} и т. д.

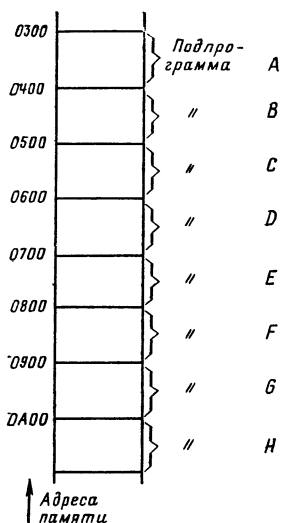


Рис. 16.6.

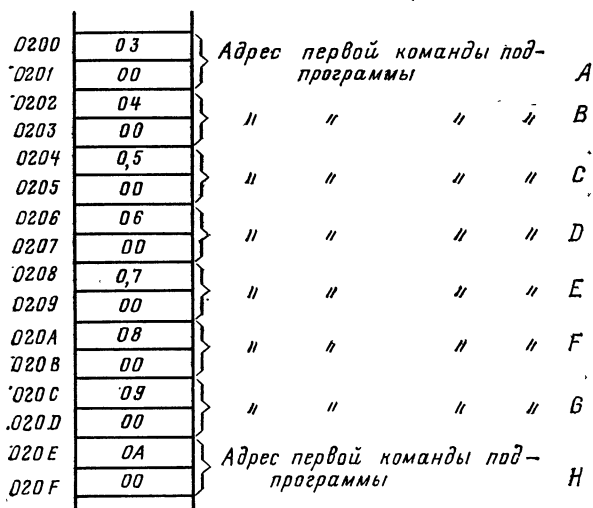


Рис. 16.7.

Входное слово с порта 2 должно быть умножено на 2 и прибавлено к числу 0200_{16} . В результате такой операции формируется адрес ячейки памяти, в которой хранится первая команда соответствующей подпрограммы. Если, как это показано на рис. 16.5, тумблеры S_1 и S_2 разомкнуты, а тумблер S_3 замкнут, то на входе порта 2 присутствует код $00000011_2 = 03_{16}$. Это означает, что должен быть выполнен переход к подпрограмме D.

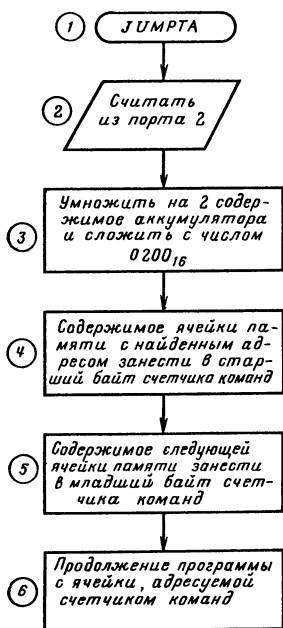


Рис. 16.8.

Блок-схема алгоритма показана на рис. 16.8. Присвоим этой программе имя JUMPTA (JUMP TABLE), так как в основе процедуры лежит операция сравнения содержимого таблицы адресов подпрограмм с кодом входного слова.

Программа приведена в табл. 16.3. Если это независимая программа, то ее следует начать с директивы ассемблера ORG, которая сообщит ассемблеру, что адрес $0000H$ ассоциируется с меткой JUMPTA.

Примечание. По команде RAL содержимое аккумулятора сдвигается на 1 разряд влево, и разряд b_0 приобретает значение признака переноса. Для получения правильного результата операции удвоения содержимого

Если умножить входное слово 03_{16} на 2 и прибавить к коду 0200_{16} , то получим $2 \cdot 03_{16} + 0200_{16} = 0206_{16}$. Первый байт адреса первой команды подпрограммы D хранится в ячейке памяти с этим адресом. Увеличение кода 0206_{16} на 1 дает адрес ячейки памяти, в которой хранится второй байт адреса. Программа должна выполнять следующие действия:

1) принять информацию со входа порта 2 и заслать ее в аккумулятор;

2) умножить на 2 содержимое аккумулятора и прибавить к числу 0200_{16} ;

3) поместить в счетчик команд содержимое ячейки памяти с полученным адресом и содержимое ячейки памяти, непосредственно следующей за ней.

аккумулятора необходимо в начале программы установить признак переноса в 0.

Т а б л и ц а 16.3

Метка	Мнемокод	Операнд	Комментарий
JUMPTA:	ORG	000H	; Адрес первой команды 000H
	IN	02H	; Ввод из порта 2 в аккумулятор
	RAL		; Умножение содержимого аккумулятора на 2
	LXI	H, 0200H	; Загрузка 0200H в регистровую пару H, L
	ADD	L	; Сложение содержимого L и аккумулятора
	MOV	L, A	; Результат отослать в регистр L
			; Адрес сформирован в регистровой паре H, L
	MOV	D, M	; Передать содержимое ячейки памяти, адресуемой регистровой парой H, L, в регистр D
	INX	H	; Инкрементировать содержимое регистровой пары H, L
	MOV	E, M	; Содержимое ячейки памяти, адресуемой регистровой парой H, L, передать в регистр E; адрес первой команды подпрограммы сформирован в регистровой паре D, E
	XCHG		; Обменять содержимое регистровых пар D, E и H, L, так как счетчик команд может обмениваться содержимым только с регистровой парой H, L
	PCHL		; Поместить содержимое регистровой пары H, L в счетчик команд

Пример 16.4. Т р е б у е т с я. Написать подпрограмму которая вырабатывает временную задержку длительностью 100 мкс.

Р е ш е н и е. Временная задержка может быть сформирована программой микро-ЭВМ, в которой некоторое множество команд не выполняет никаких операций, но занимает машинное время.

Для формирования временных задержек применяется метод, при котором в рабочий регистр загружается некоторое число и затем оно последовательно уменьшается на единицу до тех пор, пока не станет равным 0. Чем большую

задержку хотим получить, тем большее число нужно занести в этот регистр.

Блок-схема, описывающая этот метод формирования временных задержек, приведена на рис. 16.9.

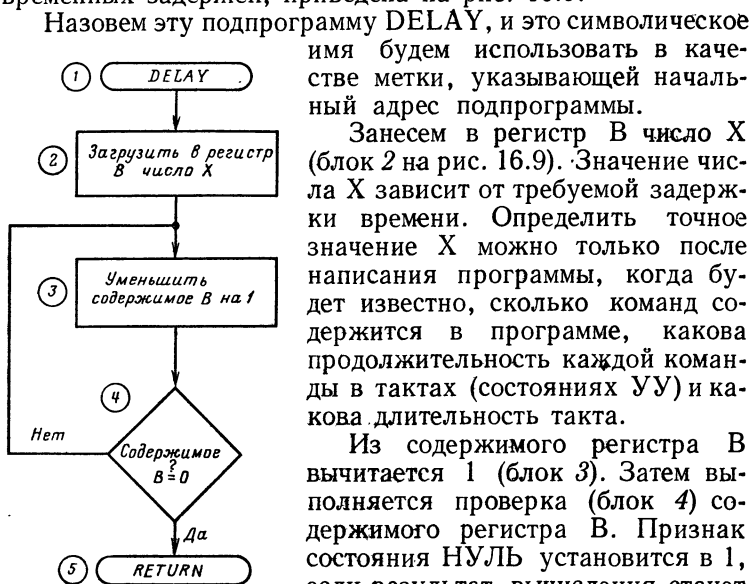


Рис. 16.9.

Из содержимого регистра В вычитается 1 (блок 3). Затем выполняется проверка (блок 4) содержимого регистра В. Признак состояния НУЛЬ установится в 1, если результат вычисления станет равным 0. Эта подпрограмма приведена в табл. 16.4.

Подпрограмма не работоспособна до тех пор, пока не будет задано значение X. Зная, какие команды используются в этой подпрограмме, можно определить время выполнения каждой команды и таким образом определить,

Таблица 16.4

Метка	Мнемокод	Операнд	Комментарий
DELAY :	MVI	B, X	; Загрузить в регистр В число X
FORWARD:	DCR	B	; Декрементировать содержимое регистра В
	JNZ	FORWARD	; Повторить процесс, если результат не равен 0; если результат нулевой, то возврат в основную программу
	RET		

сколько раз данная последовательность команд должна повторяться, чтобы получить требуемую задержку времени.

Команды MVI B, X и RET используются 1 раз. Команды DCR B и JNZ FORWARD используются многократно в зависимости от значения X.

Необходимо также учесть время выполнения команды CALL DELAY, с помощью которой программа вызывает подпрограмму DELAY.

Обратимся к системе команд (см. приложение), чтобы определить, за сколько состояний УУ выполняется каждая команда. Допустим, что каждое состояние УУ имеет длительность 500 нс, или 0,5 мкс. Получим следующие значения:

CALL	DELAY	= 17 состояний = 8,5 мкс
MVI	B, X	= 7 состояний = 3,5 мкс
DCR	B	= 5 состояний = 2,5 мкс
JNZ	FORWARD	= 10 состояний = 5,0 мкс
RET		= 10 состояний = 5,0 мкс

Команды, которые выполняются 1 раз, даются $8,5 + 3,5 + 5 = 17$ мкс.

Чтобы получить задержку времени, равную 100 мкс, микро-ЭВМ должна выполнить команды DCR B и JNZ FORWARD столько раз, чтобы этот процесс выполнялся за $100 - 17 = 83$ мкс.

Время выполнения этих команд равно $2,5 + 5,0 = 7,5$ мкс. Задержку 83 мкс нельзя получить повторением этой пары команд, так как 83 не делится на 7,5. Можно решить эту задачу, выполнив команды 10 раз. Таким образом, число X будет равно 10, и это даст задержку 75 мкс,

Т а б л и ц а 16.5

Метка	Мнемокод	Операнд	Комментарий
DELAY	MVI	B, 10	; Загрузить в регистр В число 10
FORWARD	DCR	B	; Декрементировать содержимое
	JNZ	FORWARD	; регистра В
			; Повторять процесс, если резуль-
			; тат не равен 0
	NOP		
	NOP		
	NOP		; Пустая операция
	NOP		
	RET		; Возврат в основную программу

т. е. на 8 мкс меньше, чем требуется ($83 - 75 = 8$ мкс). Можно добавить эти 8 мкс, четырехкратно используя команду NOP (No OPERATION), длительность которой равна 2,0 мкс.

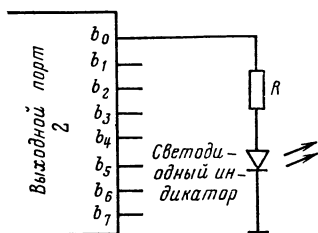


Рис. 16.10.

Окончательно подпрограмма примет вид, показанный в табл. 16.5.

Пример 16.5. Дано. Светодиодный индикатор соединен с выходным портом 2 (рис. 16.10). Светодиод загорается, если $b_0 = 1$.

Т р е б у е т с я. Написать программу включения и выключения индикатора. Частота включений не имеет принципиального значения.

Р е ш е н и е. Можно заставить индикатор зажигаться и гаснуть, формируя на выходе b_0 выходного порта 2 сигналы 1 и 0 соответственно. Занесем в аккумулятор число 01H и по команде OUT 02H передадим его в выходной порт 2. После заданной временной задержки сделаем содержимое аккумулятора равным 00H. Если снова воспользуемся командой OUT 02H, то индикатор погаснет. После заданной временной выдержки снова зашлим в аккумулятор 01H и снова отправим его содержимое в выходной порт 2. Для получения временной задержки можно использовать подпрограмму DELAY, описанную в примере 16.4. Однако требуется сформировать большую задержку времени, так как период 100 мкс слишком мал, чтобы можно было различить сигнал светодиода. Для этого надо взять большее значение X.

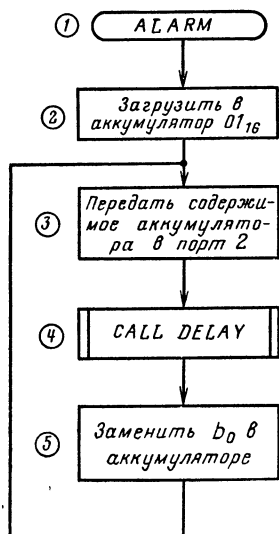


Рис. 16.11.

Блок-схема решения задачи представлена на рис. 16.11. Будем называть эту подпрограмму ALARM, используя это символическое имя в качестве метки адреса первой команды.

Загружаем в аккумулятор число $00000001_2 = 01_{16}$ (блок 2). Затем передаем это число в порт 2 (блок 3). Светодиод горит. Далее используем подпрограмму DELAY для того, чтобы светодиод горел в течение определенного времени. Подпрограмма DELAY, как отмечалось, вызывается по команде CALL DELAY (блок 4).

Изменим состояние выхода b_0 путем изменения содержимого аккумулятора (блок 5). Изменить содержимое аккумулятора можно с помощью операции ИСКЛЮЧАЮЩЕЕ ИЛИ и операнда $00000001_2 = 01_{16}$.

Содержимое аккумулятора	00000001	EXOR
Непосредственный операнд	00000001	
Новое содержимое аккумулятора	00000000	
Содержимое аккумулятора	00000000	EXOR
Непосредственный операнд	00000001	
Новое содержимое аккумулятора	00000001	

Подпрограмма приведена в табл. 16.6.

Т а б л и ц а 16.6

Метка	Мнемокод	Операнд	Комментарий
	ORG	0000H	; Первую команду поместить по адресу
			; 0000H
ALARM:	MVI	A, 01H	; Загрузить в аккумулятор 01H
FLASH:	OUT	02H	; Содержимое аккумулятора выдать в
			; порт 2
	CALL	DELAY	; Ожидание
	XRI	01H	; Изменить значение B_0 в аккумуляторе
	JMP	FLASH	; Переход

Глава семнадцатая

КОНТРОЛЛЕР СВЕТОФОРА

ДЛЯ УПРАВЛЕНИЯ ДВИЖЕНИЕМ ТРАНСПОРТА

17.1. Введение

Имеются три способа регулирования дорожного движения.

1. *Регулировщик.* Регулировщик на своем посту может учитывать интенсивность движения в любом направлении. Однако из-за большого числа перекрестков решение на сегодняшний день следует считать неудачным.

2. *Дорожные светофоры с фиксированным циклом переключения.* С помощью реле и логических схем можно построить систему управления уличными светофорами с фиксированными интервалами включения красного и зеленого света. Однако из-за отсутствия возможности анализа интенсивности движения это решение не может обеспечить высокую пропускную способность транспорта на перекрестке.

3. *Дорожные светофоры, работающие в зависимости от интенсивности движения.* С помощью детекторов, укрытых в дорожном покрытии, можно создать систему управления светофорами, которые работают в зависимости от интенсивности движения. Такой системе потребуется более сложная логика, чем у светофоров с фиксированным циклом переключения, так как при этом необходимо реализовать значительное число функций по анализу плотности потока транспорта.

С появлением МП этот способ управления дорожным движением становится самым эффективным с экономической точки зрения.

В этой главе рассмотрим систему управления дорожным движением, использующую микропроцессор. Рассмотрим следующие этапы разработки такой системы:

- 1) анализ характеристик системы;
- 2) разработка блок-схемы алгоритма;
- 3) разработка интерфейса микро-ЭВМ и внешних устройств (параллельно с этим будем вести анализ характеристик работы системы);
- 4) написание программы;
- 5) комплексирование аппаратных средств и программного обеспечения системы.

За основу возьмем микропроцессор Intel 8080, который вместе с интерфейсом и устройствами ввода-вывода должен выполнять следующие функции:

- 1) принимать сигналы от детекторов проходящего транспорта;
- 2) принимать решения о переключениях светофора на основе анализа получаемых сигналов;
- 3) переключать светофор.

17.2. Анализ характеристик системы

Постановка задачи выглядит следующим образом. Необходимо сконструировать систему, управляющую дорожным движением, для перекрестка главной и второстепенной

дорог. Управление светофором должно быть зависимым от интенсивности транспортных потоков. Сигналы о наличии транспортных средств на перекрестке поступают от детекторов, укрытых в мостовой. Места их расположения показаны на рис. 17.1. Светофоры управляются сигналами от микропроцессорного контроллера, работающего по соответствующей программе.

В зависимости от ситуации на перекрестке контроллер светофора должен обеспечить следующие режимы работы светофора.

1. Сигнал светофора для главной дороги должен быть зеленым, когда с второстепенной дороги в течение последних 30 с поступают сигналы об отсутствии транспортных средств.

2. Сигнал светофора на главной дороге должен быть зеленым в течение по крайней мере 30 с.

3. Когда с второстепенной дороги поступает сигнал о наличии транспорта, светофор для главной дороги должен немедленно включить красный сигнал. Если минимальное время зеленого сигнала еще не истекло (30 с), то система должна ждать, когда это время истечет.

4. Для второстепенной дороги включается на 30 с зеленый сигнал светофора, если от детекторов транспорта с главной дороги не поступает сигнал о наличии семи автомобилей. Сигналы от детекторов транспорта с главной дороги анализируются системой только при включенном зеленом сигнале светофора для второстепенной дороги.

5. Для обеих дорог период включения желтого сигнала светофора составляет 4 с.

6. Сигналы от детекторов игнорируются системой, когда для этой дороги включен зеленый сигнал светофора.

Проиллюстрируем на примере эти требования к системе управления светофором. Предположим, что за некоторый период времени сигналы от детекторов о движении транспорта на главной и второстепенной дорогах выглядят так, как показано на рис. 17.2, а.

Предположим, что в момент $T = 0$ с на второстепенной дороге не было автомобилей в течение последних 30 с, так что в момент $T = 0$ с на главной дороге горит зеленый сигнал светофора, а на второстепенной — красный (рис. 17.2, б).

В момент $T = 6$ с поступает сигнал о наличии автомобиля на второстепенной дороге. Сигнал светофора для главной дороги изменится на желтый и через 4 с станет

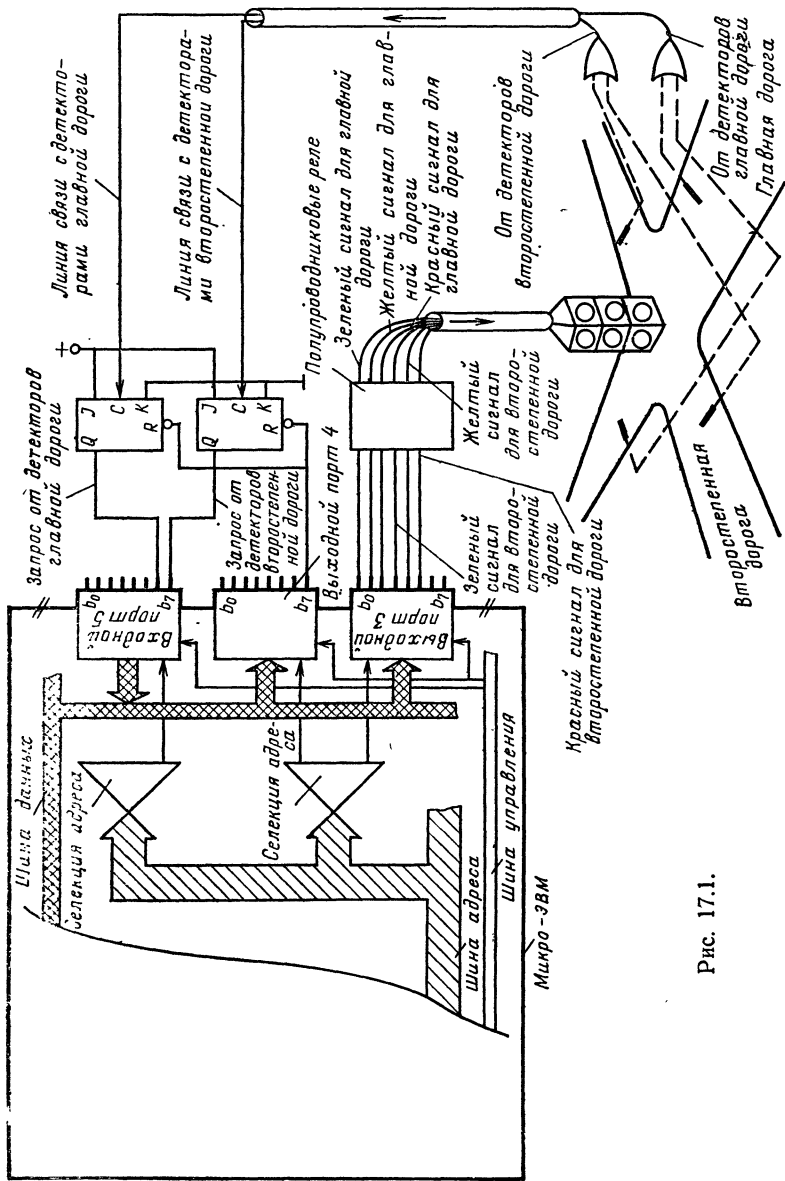


Рис. 17.1.

красным. В это же время сигнал светофора для второстепенной дороги станет зеленым. Сигнал на главной дороге останется красным до тех пор, пока не истекнут 30 с или пока от детекторов не поступит сигнал о наличии семи автомобилей на главной дороге. Сигналы от детекторов на главной дороге воспринимаются системой только в том случае, когда для главной дороги включен красный сигнал светофора (рис. 17.2, б).

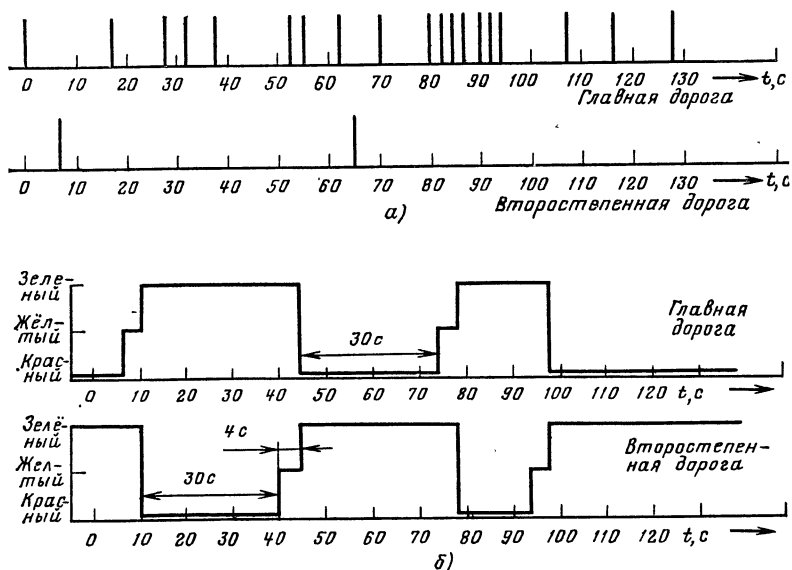


Рис. 17.2.

Однако причина, по которой в момент $T = 44$ с сигнал на главной дороге изменится на зеленый, в том, что истек максимальный период, в течение которого на второстепенной дороге может гореть зеленый сигнал светофора. В момент времени $T = 65$ с с второстепенной дороги поступает сигнал о наличии автомобиля. В момент $T = 74$ с истекает период минимального горения зеленого сигнала светофора на главной дороге и сигнал на 4 с изменится на желтый, а затем станет красным.

В момент $T = 78$ с сигнал на второстепенной дороге станет зеленым. Он останется зеленым в течение 30 с или до тех пор, пока с главной дороги не поступит сигнал о на-

личии семи автомобилей. В момент $T = 94$ с с главной дороги поступает сигнал о появлении седьмого автомобиля. В этот момент времени сигнал светофора для второстепенной дороги изменится на желтый и останется таким 4 с. В момент $T = 98$ с сигнал светофора на второстепенной дороге станет красным, а на главной зеленым.

17.3. Разработка блок-схемы алгоритма

На данном этапе разработки контроллера светофора сначала строится диаграмма состояний, в соответствии с которой должна функционировать система. Диаграмма состояний описывает работу системы при воздействии внеш-

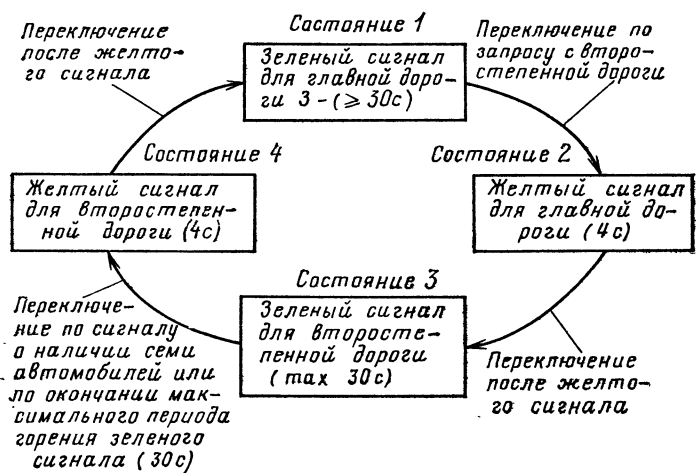


Рис. 17.3.

них сигналов детекторов и внутренних сигналов временных интервалов. Диаграмма состояний используется при построении основной блок-схемы алгоритма.

Диаграмма состояний на рис. 17.3 показывает направления переходов, которые возможны в каждом состоянии системы. Из рис. 17.3 можно увидеть условия, при которых выполняются переходы от одного состояния системы к другому.

В состоянии 1 (зеленый сигнал на главной улице) программа ожидает истечения 30-секундного интервала времени. Далее должна производиться проверка сигнала за-

проса проезда перекрестка по второстепенной дороге от детекторов, которые сообщают о наличии на ней автомобилей. Если такой запрос имеет место, то зеленый сигнал светофора на главной дороге должен измениться. Если запроса нет, то микропроцессор ждет его появления.

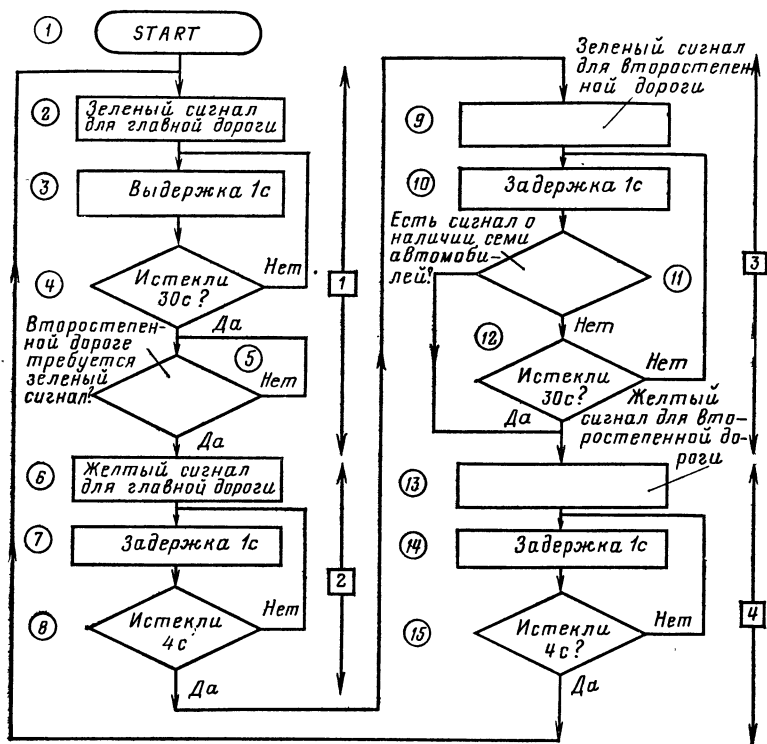


Рис. 17.4.

В состоянии 3 (желтый сигнал на главной дороге) программа выполняет вычисление интервала времени, в течение которого включен желтый сигнал светофора. Переход из состояния 3 (зеленый сигнал на второстепенной дороге) в состояние 4 (желтый сигнал на второстепенной дороге) происходит тогда, когда имеется одно из следующих условий:

- 1) истекло максимальное время 30 с для состояния 3;
- 2) с главной дороги получено требование зеленого сигнала светофора в связи с наличием семи автомобилей.

Для состояния 4 (желтый сигнал на второстепенной дороге) верны те же условия, что и для состояния 2.

На основе диаграммы состояний составляем общую блок-схему алгоритма, приведенную на рис. 17.4. Нумерация фрагментов блок-схемы (от 1 до 4) соответствует состояниям диаграммы, показанной на рис. 17.3.

17.4. Разработка интерфейса

Под интерфейсом понимаем систему связей между входным оборудованием (детекторами) и микро-ЭВМ, а также между микро-ЭВМ и выходными устройствами (исполнительными механизмами).

Как показано на рис. 17.1, входное оборудование состоит из некоторого числа детекторов автомобилей, а выходное — из электроламп светофора. Интерфейс между микро-ЭВМ и внешним оборудованием на рис. 17.1 состоит из двух триггеров, двух схем ИЛИ и шести полупроводниковых реле. При разработке интерфейса системы управления дорожным светофором возникают следующие вопросы:

1) что является выходным сигналом детектора и какой сигнал необходим для микро-ЭВМ;

2) что является выходным сигналом микро-ЭВМ и какой сигнал необходим электролампам светофора;

3) сигнал, генерируемый детектором, должен быть сохранен до тех пор, пока микро-ЭВМ под управлением программы не считывает его значение.

17.5. Интерфейс микро-ЭВМ и выходного оборудования

В светофоре для каждого направления движения используется по шесть электроламп (на лицевой и оборотной панелях светофора). Эти электролампы включены параллельно. Все эти 12 электроламп управляются 6-разрядным выходным кодом. Выходные сигналы имеют уровень сигналов ТТЛ, и поэтому электролампы не могут ими управляться. Из-за того, что нельзя объединить напряжение сети с напряжением питания микро-ЭВМ, необходимо ввести электрическую изоляцию между ним и микро-ЭВМ. Переключатель, используемый для этой цели, — это полупроводниковое реле, при помощи которого ТТЛ-сигнал от микро-ЭВМ может приводить в действие переключатель с оптоэлектронной развязкой. Полупроводниковые реле управляются непосредственно сигналами с выходного порта

микро-ЭВМ. С помощью команды вывода информации микро-ЭВМ обеспечивает включение требуемой комбинации электроламп.

17.6. Интерфейс микро-ЭВМ и системы датчиков

Предположим, что используются петлевые детекторы индукционного типа. Петли детекторов уложены под дорожным покрытием, которое образует часть колебательного контура. Когда автомобиль проходит над петлей, колебательный контур возбуждается и вырабатывается импульс. Имеются четыре детектора, по одному на каждое направление движения. Сигналы детекторов с главной дороги объединены через схему ИЛИ. То же сделано и для второстепенной дороги. Необходимо, чтобы эти два сигнала были переданы в микро-ЭВМ.

Сигналы с главной дороги должны суммироваться на счетчике по модулю 7, а сигналы с второстепенной дороги должны использоваться для формирования запроса на включение зеленого сигнала светофора для этой дороги.

Возникает вопрос, каким образом эти сигналы передавать в микро-ЭВМ. Возможно необходимо использовать внешний счетчик, который периодически опрашивается микро-ЭВМ (не достиг ли он состояния 7), или может быть микро-ЭВМ должна обрабатывать сигналы, поступающие прямо от детекторов, или сигналы должны временно запоминаться в буферном устройстве (на триггерах).

Решение этих вопросов зависит от таких факторов, как быстродействие ЭВМ и структура программы.

Если контроллеру необходимо выполнить большое число вычислений, то для операций ввода-вывода остается мало времени, и в результате этого сигнал от детектора может быть потерян.

Чтобы устранить эти потери, необходимо импульсы от детекторов запоминать в триггере. Содержимое триггера может быть опрошено, когда это требуется по программе. После этого триггер может быть сброшен при помощи сигнала, вырабатываемого микро-ЭВМ. Таким образом, реализуется периодическая проверка состояния дорожного движения под управлением программы.

Примечание. Можно было бы для этих целей использовать механизм прерываний в микро-ЭВМ, но в данном примере это усложнило бы решение.

Опрос буферного триггера производится через входной порт, который передает данные в аккумулятор, где над ними выполняются логические и арифметические операции. Входной порт вводит информацию только тогда, когда имеет место сигнал разрешения ввода. Разрешение ввода — это сигнал, который вырабатывается микро-ЭВМ, когда в программе встречается команда ввода. Адрес в этой команде задает номер порта.

17.7. Написание программы

Блок-схема программы контроллера дорожного светофора показана на рис. 17.4. В табл. 17.1 приведена программа на языке ассемблера.

Комментарий ассемблерной команды начинается после точки с запятой и служит для пояснения программы. Комментарий игнорируется в процессе ассемблирования программы. Текст, помещенный между линиями из точек, служит только для того, чтобы информировать читателя об основных решениях, принятых программистом при написании программы.

Директивы ассемблера. С помощью директивы ассемблера **ORG** предписываем ассемблирующей программе место расположения в области адресов памяти первой команды программы. Ассемблирующей программе необходима эта информация для того, чтобы разместить программу в памяти и присвоить действительные адреса символическим меткам.

Чтобы включать и выключать электролампы светофора, на выходах порта *З* требуется сформировать специальные комбинации 1 и 0 для каждого из четырех возможных состояний процесса. Чтобы не было необходимости включать в программу сложные вычисления этих выходных управляющих кодов, обозначим каждое из состояний символическим именем и поставим им в соответствие двоичные коды. Ниже приводится список символических имен состояний:

MRGR (Main Road Green) — зеленый сигнал для главной дороги, красный — для второстепенной;

MRAM (Main Road AMber) — желтый сигнал для главной дороги, красный — для второстепенной;

SRGR (Side Road GReen) — зеленый сигнал для второстепенной дороги, красный — для главной;

Программа контроллера светофора

```

.....
; назначение сигналов на выходах порта 3
; БИТ 0: зеленый на главной дороге
; БИТ 1: желтый на главной дороге
; БИТ 2: красный на главной дороге
; БИТ 3: зеленый на второстепенной дороге
; БИТ 4: желтый на второстепенной дороге
; БИТ 5: красный на второстепенной дороге
;
; Ввод сигналов от детекторов через порт 5
; Сигнал от детекторов главной дороги — бит 6
; Сигнал от детекторов второстепенной дороги — бит 7
; Сброс триггеров фиксации запросов сигналом 0 в разряде 7 порта 4
; .....

MRGR:  ORG  0000H
      EQU  21H      ; Код включения зеленого сигнала на
                   ; главной дороге
MRAM:  EQU  22H      ; Код включения желтого сигнала на
                   ; главной дороге
SRGR:  EQU  0CH      ; Код включения зеленого сигнала на
                   ; второстепенной дороге
SRAM:  EQU  14H      ; Код включения желтого сигнала на
                   ; второстепенной дороге
INIT:  CALL  RESET   ; Инициализация системы
BEGIN: MVI   A, MRGR ;
      OUT   03H      ; Выдача сигналов управления
      MVI   D, 30     ; Счетчик задержки на 30 с
LOOP1: CALL  ONESEC   ; Вызов подпрограммы задержки на 1 с
      DCR   D         ; Декрементирование счетчика
      JNZ   LOOP1     ; Если не 0, то переход к LOOP1
REQUE: IN    05H      ; Ввод данных с детекторов
      RAL          ; Сдвиг сигнала от детектора в разряд
                   ; признака переноса
      JNC   REQUE     ; Ожидание запроса с второстепенной
                   ; дороги
                   ; Сброс триггеров запросов
      CALL  RESET
      MVI   A, MRGR
      OUT   03H      ; Вывод управляющих сигналов
      MVI   D, 4      ; Счетчик задержки на 4 с
LOOP2: CALL  ONESEC   ; Вызов подпрограммы задержки на 1 с
      DCR   D         ; Декрементирование счетчика
      JNZ   LOOP2     ; Если не 0, то переход к LOOP2
      MVI   A, SRGR
      OUT   03H      ; Выдача управляющих сигналов
      MVI   E, 00     ; Сброс счетчика автомобилей
      MVI   D, 30     ; Загрузка в счетчик секунд числа 30
LOOP3: CALL  ONESEC   ; Вызов подпрограммы задержки на 1 с
      ANA   A         ; Сброс признака 0
      CALL  COUNTER   ; Вызов подпрограммы главной дороги
      JZ    FURTH     ; Если семь автомобилей, то перейти
                   ; к FURTH

```


Продолжение табл. 17.1

```

        DCR      D      ; Декрементирование счетчика
        JNZ      LOOP3  ; Если не 0, то перейти к LOOP3
FURTH:  MVI      A, SRAM
        OUT      03H    ; Выдача управляющих сигналов
        MVI      D, 4    ; Счетчик на 4 с
LOOP4:  CALL     ONESEC  ; Вызов подпрограммы задержки на 1 с
        DCR      D      ; Декрементирование счетчика
        JNZ      LOOP4  ; Если не 0, то перейти к LOOP4
JMP     BEGIN      ; Возврат к началу программы для
                     ; следующего цикла работы

```

.....

ПОДПРОГРАММА ИНИЦИАЛИЗАЦИИ

```

RESET:  MVI      A, 00H
        OUT      04H    ; Сбросить триггеры запросов
        MVI      A, 80H
        OUT      04H    ; Убрать сигналы сброса триггеров
        RET          ; Возврат в программу

```

ПОДПРОГРАММА ЗАДЕРЖКИ НА 1 С

```

ONESEC: MVI      B, FFH  ; Внешний цикл счетчика
LOOP1:  MVI      C, FBH  ; Внутренний цикл счетчика
LOOP2:  NOP          ; Пустые операции для точной вы-
                     ; держки
        NOP
        NOP
        NOP
        DCR      C      ; Декрементирование счетчика вну-
                     ; треннего цикла
        JNZ      LOOP2
        DCR      B      ; Декрементирование счетчика внеш-
                     ; него цикла
        JNZ      LOOP1
        RET

```

**ПОДПРОГРАММА ПОДСЧЕТА СЕМИ АВТОМОБИЛЕЙ
НА ГЛАВНОЙ ДОРОГЕ**

```

COUNTER: IN      5      ; Ввод данных от детекторов
        RAL
        RAL          ; Селектирование сигнала от детекто-
                     ; ров главной дороги
        JC      INCRM  ; Если есть запрос, то перейти
                     ; к INCRM
        RET          ; Возврат в программу
INCRM:  CALL     RESET  ; Сброс триггеров запроса
        INR      E      ; Инкрементировать счетчик автомо-
                     ; билей
        MVI      A, 7
        CMP      E      ; Определить наличие семи автомоби-
                     ; лей
        RET          ; Возврат в программу

```

SRAM (Side Road AMber) — желтый сигнал для второстепенной дороги, красный — для главной.

Чтобы указать, какая именно электролампа должна гореть, необходимо сформировать сигнал 1 на соответствующем выходе порта 3.

В состоянии MRGR для главной дороги должен быть включен зеленый сигнал светофора, а для второстепенной — красный. Это соответствует тому, что на выходах b_0 и b_5 выходного порта 3 должны быть 1, т. е. в состоянии MRGR на выходах выходного порта 3 должен быть сформирован код $0010001_2 = 21_{16}$.

Остальные состояния процесса кодируются следующим образом:

$$\text{MRAM} = 00100010_2 = 22_{16}$$

$$\text{SRGR} = 00001100_2 = 6C_{16}$$

$$\text{SRAM} = 00010100_2 = 14_{16}$$

Эти константы сообщают ассемблирующей программе при помощи директив EQU. В результате этого каждый раз, когда ассемблирующая программа встретит в тексте прикладной программы это символическое имя, она подставит вместо него соответствующее число.

Подпрограмма инициализации. Программа начинается с инициирования детекторных триггеров, т. е. с установки их в начальные состояния. Так как эти состояния нулевые, то будем использовать для инициирования программу установки в исходное состояние. Эта подпрограмма вызывается по команде CALL RESET.

Входы гашения детекторных триггеров соединены с выходом b_7 выходного порта 4. Сигнал гашения является действующим, когда он приобретает значение 0.

Если занесем число 00_{16} в аккумулятор и по команде OUT передадим его в порт 4, то триггеры будут установлены в исходные состояния. Сигнал гашения триггеров должен теперь приобрести недействующее значение 1, что и выполняется путем загрузки в аккумулятор числа 80_{16} и передачи его по команде OUT в порт 4.

Подпрограмма задержки на 1 с. Интервал времени 1 с формируется с помощью подпрограммы ONESEC. Здесь в регистровую пару BC загружается число FFFB, которое последовательно декрементируется до тех пор, пока содержимое регистровой пары не станет нулевым.

Команды NOP в подпрограмме используются как „тонкая доводка“ формируемой временной выдержки 1 с.

Задержка на 30 с. Время задержки 30 с получается в программе путем 30-кратного прогона подпрограммы ONESEC. Регистр D осуществляет контроль над тем, сколько раз прогоняется эта подпрограмма. Число 30 заносится в регистр D, а затем последовательно его содержимое уменьшается на 1 до тех пор, пока не станет равно 0.

На первый взгляд такой способ получения задержки 30 с кажется несовершенным, так как (см. гл. 16) эта задержка может быть получена в один прием путем занесения в регистровую пару такого числа и его последующего декрементирования, и можно сформировать задержку 30 с прежде, чем обнулится регистровая пара. Однако путем 30-кратного прогона подпрограммы задержки на 1 с можно проверять состояние детекторов на главной дороге каждую секунду.

Проверка запросов зеленого сигнала светофора для второстепенной дороги. Проверка запросов зеленого сигнала светофора для второстепенной дороги осуществляется следующим образом. Во-первых, по команде IN 5 данные с входного порта 5 пересылаются в аккумулятор. Затем по команде RAL содержимое аккумулятора сдвигается на один разряд влево. Так как данные от детекторов со второстепенной дороги поступают на вход b_7 входного порта 5, сдвиг влево на один разряд переместит эту информацию (1 или 0) в триггер переноса регистра признаков. Если содержимое разряда переноса регистра признаков 0, то это значит, что с второстепенной дороги никаких требований не поступало, и эта процедура может быть повторена. Если содержимое разряда переноса регистра признаков 1, то это означает, что со второстепенной дороги получен запрос зеленого сигнала светофора и программа должна теперь выполнить управление светофором для состояния MRAM. Этот условный переход осуществляется по команде JNC REQ.

Анализ движения на главной дороге. Переход от состояния 3 (SRGR) к состоянию 4 (SRAM) происходит или когда истекли 30 с, или когда на главной дороге детекторы зафиксировали появление семи автомобилей.

Детектирование на главной дороге обчитывается подпрограммой с символическим именем COUNT.

С детектора информация считывается по команде IN 05H и заносится в аккумулятор. Когда содержимое аккумулятора сдвигается на два разряда влево, информация, которая первоначально появилась на входе b_6 входного порта 5,

поступает в разряд переноса регистра признаков. По содержимому разряда переноса регистра признаков микро-ЭВМ может выявиться, имеется или нет на главной дороге автомобиля. Когда поступает сигнал о наличии автомобиля, триггер переноса устанавливается в 1 и в результате команды JC INCRM осуществляется переход к адресу памяти с меткой INCRM. Триггеры детекторов устанавливаются в первоначальное состояние с помощью подпрограмм RESET, и содержимое регистра E увеличивается на 1.

После выполнения этой команды число 7 перемещается в аккумулятор и сравнивается с содержимым регистра E. Если оно не равно 7, после выполнения команды CMP E признак нуля регистра признаков не устанавливается и в соответствии с командой JZ FURTH переход к состоянию 4 не происходит.

Если при вводе информации от детектора обнаруживается, что автомобили на главной дороге отсутствуют, то триггер переноса не устанавливается в 1 и в соответствии с командой JC INCRM программный переход не выполняется, а происходит возврат к основной программе. Если после возврата из подпрограммы COUNT оказывается, что обнаружено наличие семи автомобилей на главной дороге, то триггер нуля регистра признаков устанавливается в 1 и в соответствии с командой JZ FURTH осуществляется переход по программе к метке FURTH.

Если после возврата из подпрограммы COUNT наличие семи автомобилей на главной дороге не детектируется, то признак нуля не устанавливается в 1 и условие «переход по нулю» не выполняется. Содержимое регистра D уменьшается на 1 и осуществляется возврат к метке LOOP3 при условии, что содержимое регистра D не равно 0, т. е. время задержки 30 с еще не истекло.

17.8. Комплексирование аппаратных средств и программного обеспечения

Последний этап разработки системы — это слияние аппаратных средств и программного обеспечения системы. Программа, которая уже отлажена, загружается в ППЗУ системы. Остается только собрать систему и убедиться в ее работоспособности.

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

18.1. Введение

Если требуется выполнить на ЭВМ некоторую обработку данных, следует написать программу. Программа, которая может быть написана на языке ассемблера или на языке ПЛ/М, называется исходной программой. Исходная программа должна быть оттранслирована в объектную, которая затем должна быть скорректирована и протестирована. Скорректированная и проверенная программа называется *рабочей программой* ЭВМ. Исходная, объектная и рабочая

программы называются общим термином *программное обеспечение пользователя*.

Чтобы можно было оттранслировать исходную программу в объектную, проверить и скорректировать объектную программу и ввести рабочую, изготовитель МП или микро-ЭВМ предоставляет пользователю набор готовых программ. Эти так называемые вспомогательные (служебные)



Рис. 18.1.

программы принадлежат системе и имеют общее название *системного программного обеспечения*. Системное программное обеспечение не зависит от конкретного применения. Программное обеспечение пользователя и системное программное обеспечение образуют вместе программное обеспечение микро-ЭВМ. Структура программного обеспечения показана на рис. 18.1.

Системное программное обеспечение можно подразделить на резидентное программное обеспечение и кросс-программное обеспечение. Резидентное программное обеспечение включает в себя те служебные программы, необходимые для разработки прикладных программ, которые реализованы в той же микро-ЭВМ, на которой будет работать прикладная программа. Кросс-программное обеспечение включает в себя служебные программы, предназначенные для создания рабочих программ с помощью ЭВМ другого

типа. В данной главе рассмотрим системное программное обеспечение.

Выводы

1. Программное обеспечение ЭВМ состоит из программного обеспечения пользователя и системного программного обеспечения.

2. Программное обеспечение пользователя разрабатывается самим пользователем.

3. Системное программное обеспечение поставляется вместе с машиной и служит для трансляции, тестирования и корректировки прикладных программ пользователя.

18.2. Программа-монитор

Программа-монитор или просто монитор представляет собой служебную программу, которая чаще всего размещается в ПЗУ, ППЗУ или в аналогичном запоминающем устройстве. Монитор состоит из основной программы и ряда подпрограмм. Задача монитора состоит в том, чтобы *управлять работой микро-ЭВМ* в процессе трансляции, тестирования, корректировки и ввода прикладных программ пользователя. Структура монитора приведена на рис. 18.2. После того, как с помощью переключателя START включено питание, *программа раскрутки* инициализирует микро-ЭВМ. Это означает, что в счетчик команд с помощью некоторой подготовительной команды засылается адрес первой команды монитора. Программа раскрутки может быть запущена только таким способом. Таким образом, это специфическая подпрограмма, которая вызывается в результате включения напряжения питания.

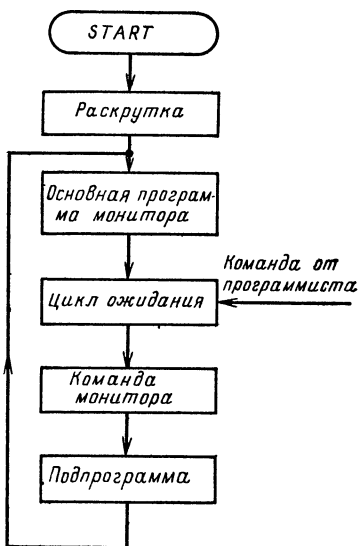


Рис. 18.2.

Основная программа монитора обеспечивает перевод монитора в цикл ожидания, в котором он остается до тех пор, пока пользователь нажатием одной или нескольких клавиш на телетайпе не запросит выполнение одной из *подпрограмм* монитора. Соответствующая подпрограмма вызывается и запускается командой монитора. С помощью этой команды монитор передает управление микро-ЭВМ вызванной подпрограмме. Монитор управляет несколькими подпрограммами. Рассмотрим две из них.

Загрузчик. Эта подпрограмма служит для управления работой устройства ввода данных с перфоленты. Загрузчик включает устройство ввода с перфоленты, обеспечивает размещение рабочей программы в нужном месте памяти и контролирует правильность расположения начального и конечного адресов. После завершения работы загрузчика управление возвращается главной программе монитора, которая переводит монитор в цикл ожидания.

Перфоратор. Эта подпрограмма обеспечивает вывод информации на перфоленту. Перфоратор имеет доступ к области памяти между начальным и конечным адресами, из которой программа выводится на перфоленту.

Выводы

1. Монитор управляет работой вычислительной системы во время трансляции, коррекции, тестирования и ввода прикладных программ пользователя.

2. Монитор состоит из основной программы, цикла ожидания и совокупности подпрограмм.

3. Когда монитор находится в режиме ожидания, пользователь может задать команду вызова подпрограммы. В этом случае монитор передает управление системой вызываемой подпрограмме.

4. Программа раскрутки загружает в счетчик команд адрес первой команды монитора.

18.3. Редактор текста

После того как составлена блок-схема, описывающая метод решения задачи, операция, задаваемая каждым блоком, может быть представлена одной или несколькими ассемблерными командами. Затем с помощью телетайпа эти команды могут быть отперфорированы на ленте. При

этом телетайп представит каждую букву, цифру или другой символ в коде ASCII.

Подготовленная таким образом перфолента называется исходной лентой.

Возникновение хотя бы одной ошибки при подготовке программы на телетайпе приводит к необходимости начи-

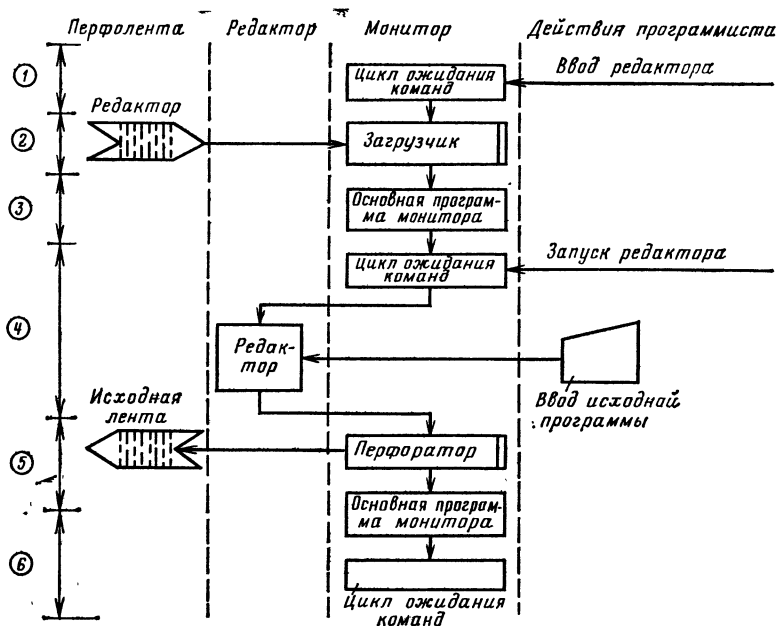


Рис. 18.3.

нать перфорацию с начала. Чтобы избежать этого, при подготовке исходной ленты используется редактор текста. *Редактор текста* или просто *редактор* позволяет корректировать и редактировать программу пользователя после того, как она нанесена на перфоленту. На рис. 18.3 показан процесс подготовки исходной ленты при помощи монитора и редактора текста.

Могут быть выполнены следующие этапы:

1. После того как программист выдал команду ВВЕСТИ РЕДАКТОР, монитор выходит из цикла ожидания.

2. Монитор вызывает подпрограмму-загрузчик, которая управляет работой устройства ввода данных с перфоленты.

3. Как только считывающее устройство ввело с перфоленты программу-редактор, монитор через свою основную программу возвращает систему в состояние ожидания следующей командой от программиста-оператора.

4. Если программист-оператор выдаст команду ЗАПУСТИТЬ РЕДАКТОР, то *из монитора* управление будет передано программе-редактору и может начаться ввод ассемблерных команд.

Когда нажимается клавиша, то соответствующая буква высвечивается на экране дисплея или печатается на теле-тайпе, а ее код (ASCII) засылается в память.

Когда введено несколько операторов (блоков) программы, программист может проверить правильность введенных символов. Если он обнаружит ошибку, то может ввести соответствующий оператор (строку) снова и повторить процедуру.

5. Когда вся программа без ошибок оказалась введенной в кодах ASCII в память, с помощью мониторной команды редактор может вызвать подпрограмму ПЕРФОРАТОР, которая управляет работой устройства вывода информации на перфоленту. Таким образом, с помощью подпрограммы перфорации программа, находящаяся в памяти, может быть выведена на перфоленту, в результате чего будет получена исходная лента.

6. После выполнения подпрограммы ПЕРФОРАТОР монитор снова переводит систему в цикл ожидания следующей команды от программиста.

Выводы

1. Редактор — это служебная программа, с помощью которой программа может быть введена символ за символом.

2. После ввода каждого символа программист имеет возможность сделать необходимые исправления.

3. Весь текст хранится в памяти.

4. Как только вся программа оказалась введенной, редактор может вывести ее на перфоленту с помощью подпрограммы ПЕРФОРАТОР.

18.4. Программа-ассемблер

Программа-ассемблер представляет собой служебную программу, которая преобразует исходную программу, написанную на языке ассемблера, в объектную. Ассемблер мо-

жет выявлять *синтаксические ошибки* в тексте исходной программы. Примером синтаксических ошибок являются: наличие или отсутствие операнда, повторное использование метки и т. п.

При трансляции (асемблировании) исходной программы ассемблер несколько раз просматривает исходный текст программы. При каждом проходе выполняется своя операция. Примерами таких операций являются:

- а) построение таблицы меток;
- б) преобразование меток в действительные адреса;
- в) замена мнемонических кодов двоичными кодами.

В процессе первого просмотра исходной программы ассемблер *формирует список* синтаксических ошибок. Асемблирование не может быть прервано для исправления ошибок. Если обнаружена ошибка, то соответствующая команда должна быть скорректирована с помощью редактора, а затем процесс асемблирования следует возобновить сначала.

На рис. 18.4 показано, каким образом программист, используя монитор, редактор и ассемблер, может преобразовать исходную программу в объектную.

Диаграмма, показанная на рис. 18.4, иллюстрирует этапы, которые должны быть выполнены, чтобы превратить исходную программу в объектную.

1. В результате выдачи команды **ВВЕСТИ РЕДАКТОР** монитор выходит из состояния ожидания и с помощью мониторной команды передает управление подпрограмме-загрузчику. Загрузчик управляет работой устройства ввода данных с перфоленты, с помощью которого в память микроЭВМ вводится редактор. Как только операция ввода закончена, монитор возвращается в режим ожидания.

2. После того как исполнена команда **ЗАПУСТИТЬ РЕДАКТОР**, можно начать ввод команд ассемблерной программы с клавиатуры. Эта операция выполняется по правилам, описанным в § 18.3. Когда завершается процесс формирования ленты с исходной программой, не содержащей ошибок, монитор возвращается в режим ожидания.

3. После этого устанавливается лента с ассемблером на устройство ввода данных с перфоленты и выдается команда **ВВЕСТИ АССЕМБЛЕР**. Под ее воздействием монитор вызывает загрузчик, который получает управление и осуществляет с помощью устройства ввода с перфоленты ввод

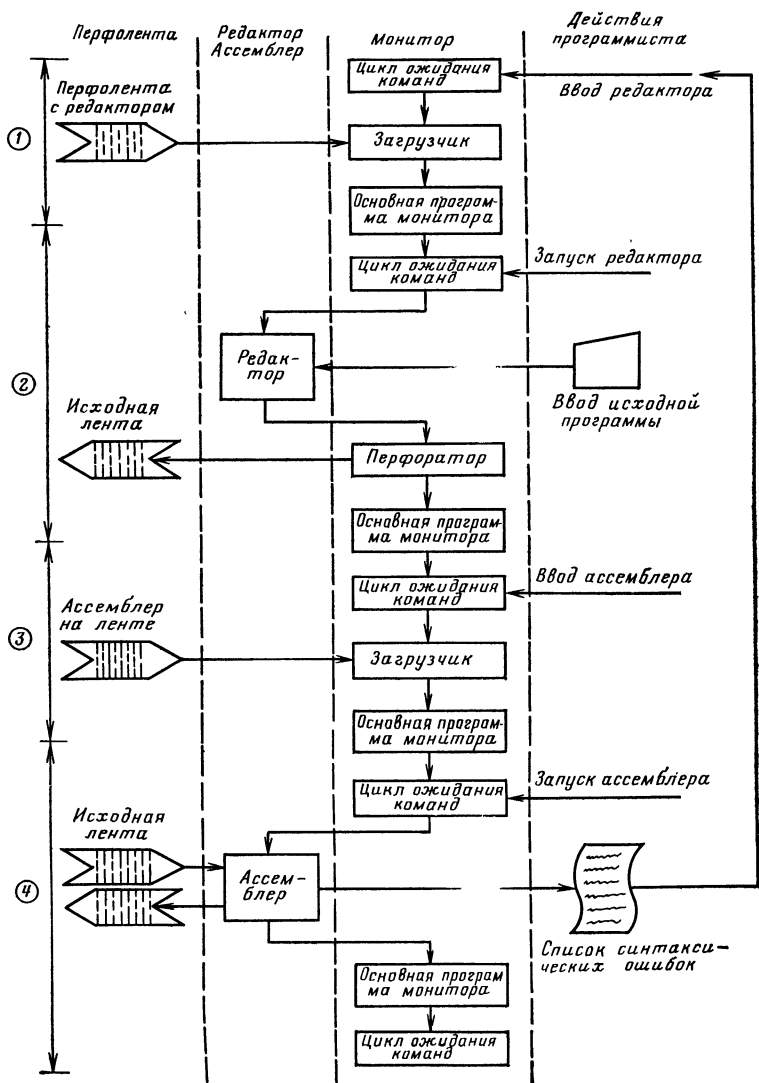


Рис. 18.4.

ассемблера в память. По завершении этой операции монитор возвращается в состояние ожидания.

4. Если наберем команду ЗАПУСТИТЬ АССЕМБЛЕР, то монитор осуществит вызов загрузчика, который через ленточное устройство ввода обеспечит ввод исходной ленты. Как только содержимое перфоленты (программа в кодах ASCII) оказалась в памяти, начинается процесс трансляции (ассемблирования).

Если в процессе трансляции ассемблер обнаружит синтаксическую ошибку, то ее можно исправить с помощью редактора. Если ассемблер в процессе работы не обнаружил ни одной ошибки и вся программа оказалась оттранслированной, то вызывается подпрограмма-перфоратор (вызов осуществляет сам ассемблер). Перфоратор управляет процессом выдачи объектной программы на перфоленту. После завершения вывода система переходит в режим ожидания.

П р и м е ч а н и е. Программа ассемблер может располагаться не только на перфоленте, но и в ПЗУ или ППЗУ. В этом случае этап ввода ассемблера, показанный на рис. 18.4, отсутствует.

Выводы

1. Ассемблер транслирует программу, написанную на языке ассемблера, в программу на машинном языке и преобразует метки в действительные адреса памяти.

2. Ассемблер формирует список обнаруженных им синтаксических ошибок, однако его работа *не может* быть прервана для их исправления.

3. Ассемблер позволяет вывести с помощью ленточного перфоратора объектную программу на перфоленту.

4. По окончании работы ассемблера монитор переводит систему в режим ожидания.

18.5. Отладчик

С помощью редактора и ассемблера получают объектную программу, не содержащую синтаксических ошибок. Однако в этой программе могут содержаться *смысловые (логические) ошибки*, вызванные разнообразными причинами:

- 1) использованием неправильных команд;
- 2) нарушением порядка команд;
- 3) некорректным резервированием ячеек памяти;
- 4) некорректным использованием меток;

- 5) смысловыми ошибками в блок-схеме;
- 6) некорректным назначением устройств ввода-вывода (например, ввод данных подключен к порту 3, а в программе встречается команда IN 04).

Если бы объектная программа сразу была введена в ЭВМ, то с большой вероятностью она, либо дала неправильный результат, либо остановилась, не закончив работы, либо заиклилась. Прежде чем запускать микро-ЭВМ для выполнения прикладной задачи, следует протестировать объектную программу. Эта работа выполняется с помощью программы отладки, или просто отладчика. Отладчик представляет собой служебную программу, обеспечивающую следующие возможности:

- 1) индикацию содержимого ячеек памяти. Введя с клавиатуры телетайпа требуемый адрес, можно обеспечить вывод содержимого ячейки памяти по этому адресу на экран дисплея или на печать;

- 2) изменение содержимого памяти. Содержимое ячейки памяти можно изменить, используя клавиатуру телетайпа. Программист вводит адрес и новое содержимое соответствующей ячейки памяти;

- 3) индикацию содержимого регистров. По команде программиста содержимое РОН, счетчика команд, аккумулятора, регистра адреса, регистра-указателя стека или регистра признаков может быть выдано на экран дисплея;

- 4) останов по адресу. Если указать один или несколько адресов (точек останова), то выполнение объектной программы будет прерываться при обращении к команде с соответствующим адресом. После останова программист, используя возможности индикации содержимого памяти и регистров, может проверить правильность работы выполненной части программы. Если возникают ошибки, то их можно исправить с помощью средств изменения содержимого памяти. После выполнения нужных операций программист может продолжать работу программы, выдав команду ПРОДОЛЖИТЬ;

- 5) пошаговый режим. В этом режиме можно выполнять программу по одной команде. После выполнения каждой команды программист может проверить результат с помощью средств индикации содержимого памяти и регистров. Если будет обнаружена ошибка, то ее можно исправить средствами изменения содержимого памяти. Разновидностью пошагового режима является n -тактный режим, при котором, задавая значение n , программист указывает, какое число

последовательных команд должно быть выполнено. Для пошагового режима $n = 1$. Как только исполнена одна или несколько команд (в зависимости от значения n), отладчик переходит в режим ожидания команд: ПРОДОЛЖИТЬ, ИНДИЦИРОВАТЬ СОДЕРЖИМОЕ ПАМЯТИ или ИНДИЦИРОВАТЬ СОДЕРЖИМОЕ РЕГИСТРОВ.

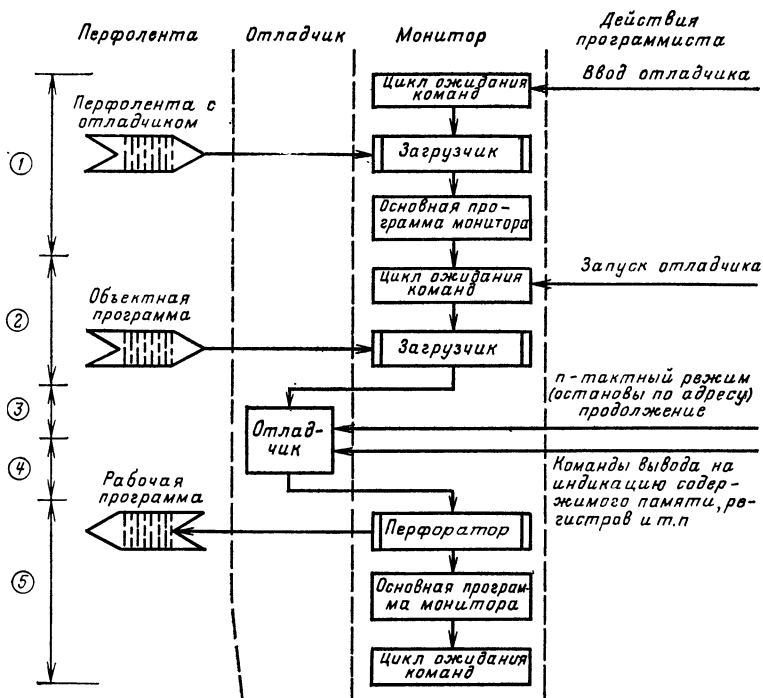


Рис. 18.5.

Процесс работы с отладчиком схематически изображен на рис. 18.5. При работе с отладчиком выполняются следующие этапы.

1. При выдаче команды ВВЕСТИ ОТЛАДЧИК монитор выходит из состояния ожидания и с помощью мониторинговой команды вызывает загрузчик. Загрузчик начинает работу и осуществляет с помощью устройства ввода с перфоленты ввод отладчика. После завершения ввода отладчика в память микро-ЭВМ монитор возвращается в режим ожидания,

2. Если выдать команду ЗАПУСТИТЬ ОТЛАДЧИК, то монитор передаст управление загрузчику, который с устройства ввода с перфоленты введет объектную программу. Объектная программа была получена в результате процесса, изображенного на рис. 18.4. После того как объектная программа введена в память микро-ЭВМ, отладчик переходит в режим ожидания.

3. Теперь с помощью телетайпа программист может задать либо конкретное значение параметра n , либо совокупность адресов (точек останова), в которых выполнение программы будет прерываться.

Затем он выдает команду ПРОДОЛЖИТЬ, и в зависимости от значения n выполняется одна или несколько команд, после чего отладчик возвращается в состояние ожидания.

4. В это время программист имеет возможность посмотреть содержимое ячеек памяти или регистров и выполнить замену их содержимого. Когда эти операции будут выполнены, отладчик вернется в режим ожидания и после выдачи команды ПРОДОЛЖИТЬ обеспечит выполнение следующей группы команд.

5. Как только отладка программы (с помощью рассмотренных средств) полностью завершена, отладчик передает управление перфоленте, который обеспечивает вывод окончательного варианта объектной программы на перфоленту. Эта отлаженная программа называется рабочей программой.

Выводы

1. Используя отладчик, можно осуществить выполнение программы по частям, что облегчит обнаружение и исправление логических ошибок.

2. Используя n -тактный режим и точки остановов, можно последовательно проверять работу небольших частей объектной программы.

3. Используя средства выдачи содержимого регистров и ячеек памяти, можно выводить их на экран дисплея для анализа.

4. С помощью команды изменения содержимого памяти можно корректировать содержимое любой ячейки памяти.

5. Выполнение каждого очередного отрезка программы инициируется командой ПРОДОЛЖИТЬ.

18.6. Имитатор

Было установлено, что с помощью таких служебных программ, как ассемблер, редактор, монитор и отладчик,

можно получить откорректированную объектную программу, т. е. рабочую программу. Оборудование, на котором происходит процесс превращения исходной программы в рабочую, называется системой разработки. Эта система не обязательно должна включать в себя те же самые типы микропроцессоров, запоминающих устройств и внешних устройств, с которыми будет работать рабочая программа. Но она должна работать точно так же.

Если работаем с несколькими различными микромашиными системами, то можно создавать рабочие программы для них с помощью одной системы разработки. При этом всякий раз системе разработки следует сообщить особенности микросистемы, для которой создается рабочая программа.

Кроме того, система разработки должна иметь внешние устройства, аналогичные разрабатываемой системе, чтобы была возможность проверить правильность выполнения команд ввода-вывода.

Различные микромашинные системы могут имитироваться на одной и той же системе разработки. Это осуществляется с помощью служебной программы, известной под названием *имитатор*. С помощью имитатора можно заставить микро-ЭВМ, построенную на базе микропроцессора 6800, работать так же, как микро-ЭВМ, построенная на микропроцессоре 8080.

Имитатор обеспечивает следующие функции:

1) имитацию характеристик конкретного типа МП.

Это означает, что система команд имитируемого МП должна быть преобразована в систему команд МП, лежащего в основе системы разработки;

2) имитацию периферийных устройств той микро-ЭВМ, для которой создается рабочая программа;

3) возможность отладки объектной программы на микро-ЭВМ иного типа, чем та микро-ЭВМ, для которой разрабатывается программа¹.

Отсюда следует, что в состав имитатора входит отладчик.

18.7. Компилятор

До сих пор предполагалось, что исходная программа написана на языке ассемблера и транслируется в объектную программу с помощью ассемблера.

¹ В качестве средства разработки и отладки рабочих программ чаще всего используются не микро-ЭВМ иного типа, а мини-ЭВМ и большие ЭВМ в кросс-режиме. (Прим. ред.)

Однако программа может быть написана на одном из языков высокого уровня, таких как АЛГОЛ, КОБОЛ, ФОРТРАН, ПЛ/М или БЕЙСИК. Служебные программы, которые осуществляют трансляцию программ, написанных на языках высокого уровня, в объектные программы, называются *компиляторами*.

По существу компилятор делает то же самое, что и ассемблер. Но из-за того, что каждому оператору на языке высокого уровня соответствует большое число команд в объектной программе, компилятор более «экстенсивен», чем ассемблер¹.

Выводы

1. Микропроцессор и внешние устройства микромашиной системы могут имитироваться в системе разработки с помощью имитатора.

2. В состав имитатора входит отладчик.

3. Компилятор транслирует программу, написанную на языке высокого уровня, в объектную.

4. Система разработки представляет собой комплекс средств, предназначенных для создания рабочих программ.

18.8. Кросс- и резидентное программное обеспечение

Принято различать резидентное и кросс-программное обеспечение. Под кросс-программным обеспечением подразумевается совокупность тех служебных программ, которые предназначены для создания рабочих программ на микро-ЭВМ, отличающейся от той, для которой предназначена рабочая программа.

К числу служебных программ, относящихся к кросс-программному обеспечению, следует отнести: имитатор, кросс-ассемблер, кросс-компилятор.

Кросс-ассемблер и кросс-компилятор — это трансляторы, с помощью которых исходная программа может быть преобразована в объектную на ЭВМ, отличающейся от микро-ЭВМ, на которой рабочая программа должна выполняться.

В состав резидентного программного обеспечения входят все служебные программы системного программного обес-

¹ В процессе компиляции объектная программа всегда получается более длинной, чем в процессе ассемблирования. (*Прим. ред.*)

печения, которые не входят в состав кросс-программных средств. К ним относятся служебные программы, которые:

1) используются при создании рабочей программы на ее собственной микромашинной системе;

2) используются, когда система разработки идентична той микромашинной системе, для которой создается рабочая программа (когда не требуется имитация);

3) являются независимыми от того, требуется или нет имитация (например, монитор, редактор, отладчик).

Служебные программы, относящиеся к резидентному программному обеспечению, следующие: ассемблер (резидентный ассемблер), компилятор (резидентный компилятор), монитор, редактор текста, отладчик.

Выводы

1. Кросс-программное обеспечение включает в себя те служебные программы, которые используются при имитации микропроцессора в системе разработки при создании для него рабочих программ.

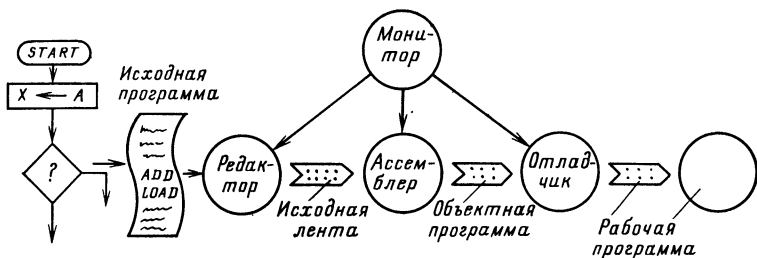


Рис. 18.6.

2. Кросс-программное обеспечение включает в себя имитатор, кросс-ассемблер и кросс-компилятор.

3. Резидентное программное обеспечение включает в себя редактор текста, монитор, отладчик, ассемблер (резидентный), и компилятор (резидентный).

4. На рис. 18.6 показан процесс подготовки рабочей программы из исходной средствами системного программного обеспечения.

Сначала разрабатывается блок-схема, которая описывает метод решения задачи.

Операция, определенная в каждом блоке, преобразуется в команды на языке ассемблера.

С помощью редактора создается исходная лента, на которой все символы (буквы, цифры и т. п.) представлены в коде ASCII.

Ассемблер преобразует исходную ленту в объектную, на которой средствами отладчика выявляются и исправляются все логические ошибки.

Окончательным результатом является рабочая программа.

Глава девятнадцатая

СИСТЕМЫ РАЗРАБОТКИ

19.1. Введение

При практической работе с микро-ЭВМ требуется не только знание электрической схемы и наличие паяльника, с которым обычно имеют дело. Чтобы устройство начало выполнять возложенные на него функции, необходимо наладить электронную аппаратуру, разработать правильное программное обеспечение.

После того, как программа размещается в ПЗУ или ППЗУ, в нее нельзя вносить никаких изменений. Поэтому в процессе разработки программы необходимо быть очень внимательным. Чтобы сделать процесс разработки более эффективным, используют *микромашинные системы разработки (МСР)*.

Система разработки представляет собой вычислительную систему, центральным элементом которой является МП и которая оснащена набором системных программ. Поскольку разрабатываемая программа ориентирована на работу с определенным типом МП, система разработки должна иметь в своем составе микропроцессор того же типа либо возможность имитировать его.

19.2. Типовой цикл разработки

Если при разработке системы используются традиционные средства типа элементов ТТЛ, то разработка осуществляется по схеме, приведенной на рис. 19.1.

1. Разработка системных спецификаций. На этом этапе формируется перечень требований к системе и определяются формы представления входных и выходных данных.

2. Разработка временных диаграмм. На этом этапе строятся диаграммы прохождения сигналов в системе и производится разбиение на функциональные блоки.

3. Разработка схем. На данной стадии производится детальная проработка структуры и конструкции функциональных блоков и вопросов их взаимодействия.

4. Тестирование схем и внесение необходимых исправлений. На этой стадии работ проверяется, насколько удовлетворяют спецификациям сконструированные схемы. Если это необходимо, то в них вносятся изменения.

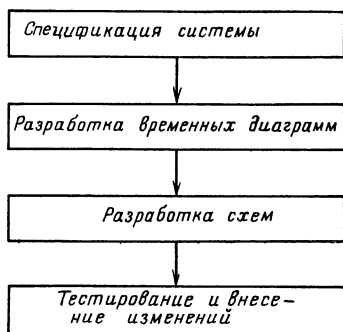


Рис. 19.1.

19.3. Цикл разработки систем на основе микропроцессоров

Разработка систем, в основе которых лежит МП, осуществляется, в сущности, по аналогичной схеме. Однако при этом добавляется несколько этапов (рис. 19.2).

1. Разделение функций аппаратных средств и программного обеспечения. Отдельные функциональные блоки, полученные в результате анализа временных диаграмм, могут, в общем случае, реализовываться как аппаратными, так и программными средствами. Выбор способа реализации в каждом конкретном случае обосновывается технико-экономически.

Пример. Если требуется реализовать временную задержку, то аппаратно ее можно осуществить, используя одновибратор. То же самое можно выполнить с помощью программного цикла. Выбор аппаратного решения имеет то преимущество, что высвобождается процессор, который может быть загружен другой работой. Программный вариант решения позволяет исключить лишние элементы из системы и дает возможность программно управлять длительностью задержки.

2. Разработка программного обеспечения. Этот этап проектирования состоит из разработки блок-схемы, выде-

ления в ней законченных частей (подпрограмм), написания программы на подходящем языке программирования и трансляции ее в объектную программу (в программу на машинном языке).

В большинстве случаев появление нового этапа разработки программного обеспечения не создает дополнительных проблем, а приводит к их возможному перемещению из области аппаратной в область программного обеспечения. Подключение МП к периферийному оборудованию может рассматриваться как обычная операция, выполнение

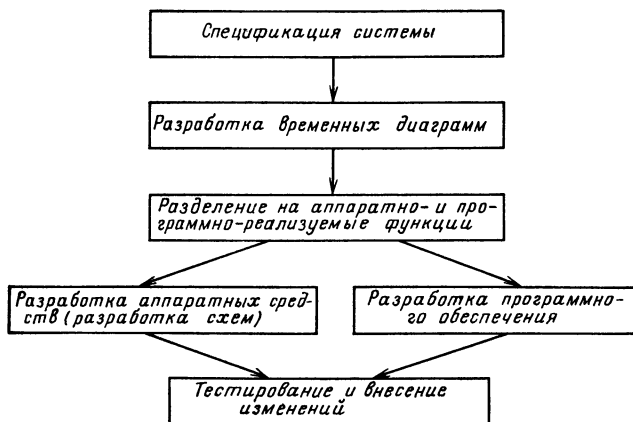


Рис. 19.2.

которой не требует особых затрат. После этого фактически и начинается вся работа по созданию системы. В больших ЭВМ эту работу выполняют проектировщики систем или системные программисты. При использовании более простого устройства — МП задача создания программного обеспечения ложится на специалиста-схемотехника. В результате этого он часто чувствует себя весьма неуютно!

Изготовители микропроцессоров прекрасно представляют себе это и стараются сделать так, чтобы схемотехник мог успешно работать, используя несложное программное обеспечение. Это достигается путем разработки обширной документации, предоставления сервисных программ, организации курсов, а также поставки средств, облегчающих процесс разработки.

Из разнообразия предлагаемых пользователям систем разработки не всегда просто выбрать наиболее подходящую.

Постараемся создать у читателя четкие представления о том, как это сделать.

Выводы

1. При разработке программ для микро-ЭВМ часто используются системы разработки.

2. Традиционная схема разработки состоит из следующих этапов:

- 1) определение системных спецификаций;
- 2) разработка временных диаграмм;
- 3) разработка схем;
- 4) тестирование схем и внесение изменений.

3. Цикл разработки при использовании МП состоит из следующих этапов:

- 1) определение системных спецификаций;
- 2) разработка временных диаграмм;
- 3) разделение программно- и аппаратно-реализуемых функций;
- 4) разработка аппаратных средств и программного обеспечения;
- 5) тестирование и внесение изменений.

19.4. Методы разработки программ

Программирование заключается в том, что составляется такая последовательность команд, которая управляет работой МП, что обеспечивает функционирование системы в соответствии с предъявляемыми требованиями. Эта прикладная программа должна быть преобразована в машинную форму и размещена на соответствующем типе носителя (перфоленте, магнитной ленте или гибком диске).

Для облегчения процесса разработки прикладных программ изготовитель предоставляет такие сервисные программы, как редактор, ассемблер, имитатор и т. п. Проблема состоит в том, что воспользоваться этими программами можно только в том случае, если имеется ЭВМ. Существует много различных вариантов доступа к вычислительным средствам:

- 1) аренда терминала, подключенного к вычислительной системе с разделением времени;
- 2) использование собственной ЭВМ;
- 3) покупка системы разработки.

19.5. Разработка программ с использованием системы разделения времени

При разработке программ с использованием системы разделения времени служебные программы, предлагаемые изготовителем, находятся в памяти большой вычислительной машины, работающей в режиме разделения времени. Арендуя телефонный канал связи, получаем доступ не только к ЭВМ, но и к служебным программам. Такая организация работ обладает следующими преимуществами:

1) отсутствием капитальных затрат. Достаточно только выделить место для арендуемого терминала (например, терминала или дисплея) и для аппаратуры связи;

2) отсутствием задержек в обслуживании. При обращении к системе с разделением времени достаточно указать имя служебной программы и можно сразу начать работу;

3) практически неограниченной емкостью памяти;

4) простотой внесения изменений в программы.

В то же время работа в системе с разделением времени не лишена недостатков:

1) достаточно высокая арендная стоимость времени ЭВМ (в зависимости от типа используемых служебных программ и требуемой емкости памяти арендная плата изменяется в пределах от 100 до 500 гульденов в час);

2) отсутствие возможности проверки правильности работы программы на том оборудовании, для которого она предназначена (приходится ограничиваться режимом имитации);

3) отсутствие возможности проверки совместной работы периферийного оборудования, ориентированного на данное применение, с разрабатываемой программой.

В результате этого системы с разделением времени используются для разработки программ в следующих случаях:

1) при сопоставлении свойств различных микропроцессорных систем;

2) когда собственная ЭВМ перегружена;

3) если имеется опытный программист;

4) когда у пользователя уже имеется доступ к терминалу.

19.6. Разработка программ на собственной ЭВМ

Пользователи, которые уже имеют свою собственную ЭВМ (начиная от PDP-11 и кончая IBM 370), проявляют

стремление разработать прикладные программы на собственной машине. При этом служебные программы должны быть приспособлены к особенностям конкретной ЭВМ. Эта процедура может потребовать от квалифицированного программиста от половины рабочего дня до целой рабочей недели, что следует учесть при исчислении соответствующих затрат. Разработка программы на собственной ЭВМ имеет следующие преимущества:

- 1) возможность использования существующего периферийного оборудования (построчного печатающего устройства, накопителя на магнитной ленте, дисплейного терминала, читающего устройства с перфокарт и выходного перфоратора), что позволяет сэкономить достаточно много времени;

- 2) возможность использования существующего программного обеспечения;

- 3) наличие большого объема памяти;

- 4) небольшие дополнительные затраты. Служебные программы могут стоить от 2000 до 10 000 гульденов. Естественно, что за все программное обеспечение (ПО), которое поставляется с микропроцессором, должен платить пользователь (либо за дорогостоящее программное обеспечение, либо за дорогостоящие аппаратные средства);

- 5) наличие на ЭВМ квалифицированного программиста, который может помочь стажеру-схемотехнику в решении проблем программирования.

Разработка программ на собственной ЭВМ не лишена недостатков:

- 1) поскольку собственная ЭВМ используется и для решения других задач, работа может часто прерываться;

- 2) при использовании программных средств имитации (которые также должны быть разработаны) тестирование разработанных программ может быть лишь частичным;

- 3) практически почти совсем невозможно или очень сложно подключить специальное периферийное оборудование, которое предполагается использовать в микропроцессорной системе, и проверить правильность его взаимодействия с разрабатываемой программой.

По этой причине прикладные программы разрабатываются на своих собственных ЭВМ только в следующих случаях:

- 1) если своя ЭВМ недогружена;

- 2) в процессе ассемблирования и редактирования прикладных программ;

3) когда предстоит разработать так много программ, что приобретение для этой цели специальной ЭВМ (например, PDP-11) может быть оправдано.

19.7. Разработка программ с использованием аппаратных средств разработки

Под системой разработки подразумевается комплекс средств, которые предназначены специально для разработки программ для микро-ЭВМ. Эти системы поставляются изготовителями микропроцессоров. В зависимости от их стоимости эти системы предлагают широкий набор средств для разработки прикладных программ. Обычно такие системы построены на том же типе МП, который будет работать в проектируемом микропроцессорном устройстве. Полученная программа без дополнительной доводки может работать в проектируемом устройстве. Из сказанного следует, что для каждого типа МП требуется своя система разработки. Приобретение конкретной системы разработки приводит покупателя к необходимости использовать в дальнейшем единственный тип МП *Caveat emptor*¹.

Система разработки обычно оснащена *резидентным программным обеспечением*, которое представляет собой совокупность программ, разработанных специально для этой системы и готовых к использованию. Тем самым оно отличается от *кросс-программного обеспечения*, которое после некоторой настройки может использоваться на различных типах собственных ЭВМ. Достоинствами резидентного программного обеспечения являются:

- 1) независимость от других ЭВМ;
- 2) невысокая стоимость (обычно включаемая в стоимость системы разработки);
- 3) возможность тестирования программы в реальном времени со всеми преимуществами, вытекающими из возможности использования периферийного оборудования, такого как АЦП, ЦАП, сигнальные лампы и т. п., которое может быть подключено к системе.

Система разработки с резидентным программным обеспечением является наиболее удобной для инженера-схематехника, который начинает заниматься программированием. Это позволяет ему работать независимо от других и практиковаться в искусстве программирования за своим собствен-

¹ Берегись, владелец! (Прим. пер.)

ным столом. Когда он делает ошибку — только он один знает об этом!

Далее рассмотрим устройство рабочего места программиста.

Выводы

1. Рабочая программа обычно создается с помощью служебных программ.

2. Служебные программы (или системное программное обеспечение) подразделяются на резидентное ПО и кросс-ПО.

3. Рабочая программа может быть создана с использованием:

- 1) коммерческой системы телеобработки;
- 2) своей собственной ЭВМ;
- 3) системы разработки.

19.8. Организация системы разработки

На рис. 19.3 показана общая структура системы разработки. Различные блоки (модули) в зависимости от их размеров размещаются обычно каждый на своей печатной плате или все на одной плате.

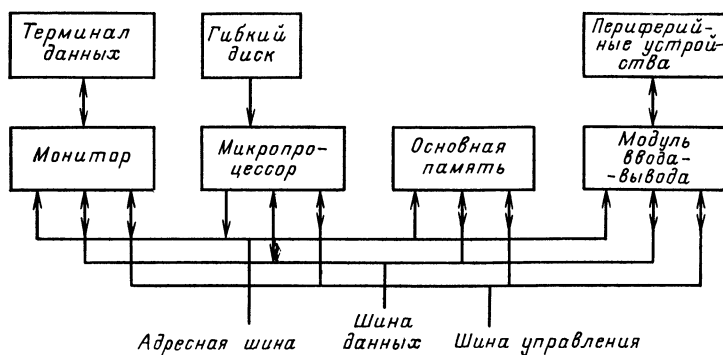


Рис. 19.3.

Микропроцессорный блок. В этот блок входит не только сам микропроцессор, но и другие компоненты, без которых МП не смог бы работать. К ним относятся, например, генератор тактовых сигналов, схемы запуска с антидребезговым устройством, ПДП, формирования прерываний и т. п.

Монитор — это „мозг“ системы разработки. Он включает в себя аппаратные средства и соответствующее ПО и служит посредником между МП и терминалом (например, теле-тайпом). Информация в любых случаях проходит через монитор, который может влиять на работу МП. Основой монитора является мониторная программа, обычно располагаемая в ППЗУ. От качества и возможностей этого „жесткого ПО“ (ПО в постоянной и неизменной форме) зависит удобство работы с системой разработки. Длина мониторной программы может лежать в пределах от нескольких сотен до нескольких тысяч слов.

Основная память обычно реализуется в виде ОЗУ или иногда в виде недорогого РППЗУ, содержимое которого может быть стерто в помощью ультрафиолетового облучения. Все служебные программы хранятся в основной памяти. Она используется также для временного хранения данных. Как только требуемое ПО разработано, служебные программы не обязательно оставлять в основной памяти. Они хранятся в ППЗУ, которое дешевле ОЗУ. Очевидно, если программа введена в ПЗУ, то нет возможности изменить ее! В МП, которые используются для решения прикладных задач, дорогостоящее ОЗУ используется только для временного хранения данных. Однако для системы разработки с ее служебными программами память емкостью от 8 К до 12 К (1 К = 1024 ячеек) не является чрезмерной. Можно работать, используя память объемом всего 1 К, но это будет работа на очень примитивном уровне: программировать придется на машинном языке (не имея возможности использования ассемблера, например).

Модуль ввода-вывода. Система разработки приобретает особую практическую ценность, если после разработки программы ее можно использовать в качестве прототипа. Для этого должна быть обеспечена возможность подключения к системе разработки всех периферийных устройств, которые будут использоваться в прикладной системе. Только в этом случае система может быть подвергнута всеобъемлющему контролю. Периферийные устройства, о которых идет речь, могут включать в себя ЦАП, АЦП, исполнительные механизмы, дисплеи, модели объектов управления и т. п. В зависимости от типа периферийного устройства передача данных осуществляется параллельно или последовательно в синхронном или асинхронном режиме. Модуль ввода-вывода является посредником между МП и периферийными устройствами, осуществляющим со-

гласование уровней электрических сигналов и скоростей передачи данных.

Модуль ввода-вывода обычно построен на ИС, которые специально разработаны для данного типа МП. Он содержит схемы шинных драйверов, дешифраторов адреса и т. п. Выходы периферийных устройств обычно совместимы со схемами ТТЛ и КМОП и снабжены линиями прерываний. С помощью линий прерываний периферийное устройство может просигнализировать о том, что оно готово к обмену данными с МП. При появлении такого сигнала МП прерывает выполняемую процедуру и ведет диалог с периферийным оборудованием. Хорошо сконструированный модуль ввода-вывода позволяет снизить затраты на создание аппаратных и программных средств прототипа.

Выводы

1. Система разработки состоит из: микропроцессора, главной памяти, модуля ввода-вывода, монитора.
2. Монитор управляет работой всей системы.
3. Монитор состоит из программной части (мониторная программа) и аппаратных средств.
4. В основной памяти временно хранятся как служебные программы, так и обрабатываемые данные.
5. Модуль ввода-вывода осуществляет согласование работы периферийных устройств с остальными частями системы разработки.

19.9. Устройства ввода-вывода

После того как завершено создание рабочей программы и в тот момент, когда эта программа выполняется на микропроцессоре, для которого она предназначена, возникает задача обеспечения взаимодействия МП с различным периферийным оборудованием типов АЦП, ЦАП, исполнительных реле, детекторов, индикаторных ламп и т. п.

При создании рабочей программы с помощью системы разработки работают с периферийными устройствами совсем иного типа (рис. 19.4).

Для большинства систем разработки характерны следующие периферийные устройства:

- 1) диалоговый терминал;
- 2) устройства ввода-вывода с перфоленты;
- 3) устройство ввода-вывода с магнитной ленты;

4) гибкий диск.

В данной главе дано краткое описание этих устройств. Более подробную информацию по периферийному оборудованию можно найти в гл. 20.

Диалоговый терминал. В качестве диалогового терминала обычно применяется телетайп или терминал с клавиатурой и видеодисплеем. Он используется в процессе разработки программы. С помощью диалогового терминала, который подключен к МП через монитор, в систему можно ввести и запустить такие служебные программы, как редактор, ассемблер и отладчик. Если, например, система управле-

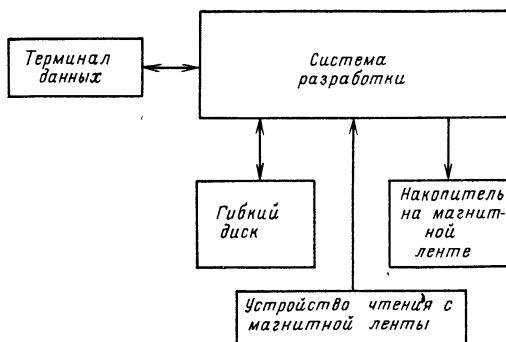


Рис. 19.4.

ния светофором дорожного движения работает неправильно, программист может с помощью диалогового терминала вывести на экран (или на печать) содержимое ячеек памяти или изменить его. Как только программа начинает работать правильно, диалоговый терминал становится ненужным. В прикладной системе диалогового терминала нет. Система разработки, в которой отсутствует диалоговый терминал, совершенно неработоспособна.

Обычно обмен информацией между системой разработки и диалоговым терминалом осуществляется в коде ASCII. Этот 7-битный код служит для представления букв и цифр, а также некоторых других служебных символов. Данные в коде ASCII вводятся в систему разработки последовательно (бит за битом), а монитор выполняет их преобразование в параллельный код.

Во многих современных терминалах используется электронно-лучевая трубка (ЭЛТ). Такой терминал имеет обычную клавиатуру, но вместо печати на бумаге вводимые

символы высвечиваются на экране ЭЛТ. Достоинством такого устройства является его бесшумность, недостатком — отсутствие „твердой копии“. В результате отсутствует возможность по окончании работы проверить выполненные операции. От этого недостатка можно избавиться, если подключить печатающее устройство или ленточный перфоратор, которые можно включать только при необходимости получения документа. Программируемый терминал представляет собой довольно дорогое устройство. Однако при использовании в качестве терминала телетайпа расходуется большое количество бумаги, поскольку печатается любой (даже самый незначительный) текст.

Устройство ввода с перфоленды и ленточный перфоратор. Программы, хранимые в ОЗУ, являются „разрушаемыми“, поскольку при отключении напряжения питания содержимое памяти уничтожается. Поэтому программы обычно хранятся на перфолентах.

Формат, в котором программа выводится на перфоленду, зависит от особенностей мониторной программы; к сожалению, стандарты отсутствуют и имеется столько форматов, сколько существует систем разработки. В одних случаях на перфоленду наносятся контрольные разряды (биты четности) для обеспечения максимальной надежности, в других — экономят именно на этих разрядах, чтобы максимально сократить длину перфоленды.

Простейшее устройство ввода данных с перфоленды и ленточный перфоратор имеются в составе телетайпа ASR 33. Они так соединены с телетайпом, что не требуется дополнительных средств их подключения к системе разработки.

Устройство ввода-вывода с магнитной ленты. Программу можно хранить на магнитной ленте. В настоящее время очень распространенными являются малогабаритные кассеты фирмы „Филипс“. При этом магнитная лента для использования в ЭВМ должна удовлетворять жестким требованиям, которым не отвечают магнитные ленты для магнитофонных записей. В составе некоторых терминалов имеются кассетные НМЛ, поставляемые как дополнительное средство. Они очень удобны в работе. Для тех, кого не устраивают кассетные НМЛ, имеется возможность использовать массовое внешнее ЗУ на гибком диске. В каждом случае выбор типа ВЗУ должен соответствовать объему хранимых данных, который существенно различен для различных систем разработки.

Накопитель на гибком диске. В принципе накопитель на гибком диске не отличается от НМД, используемых на больших ЭВМ. Особенностью гибкого диска является то, что он выполнен на гибкой пластиковой основе, на которую нанесен магнитный слой. Диск вращается с фиксированной частотой, а с помощью перемещаемых в радиальном направлении головок чтения-записи можно получить доступ к различным дорожкам. Одни системы разработки

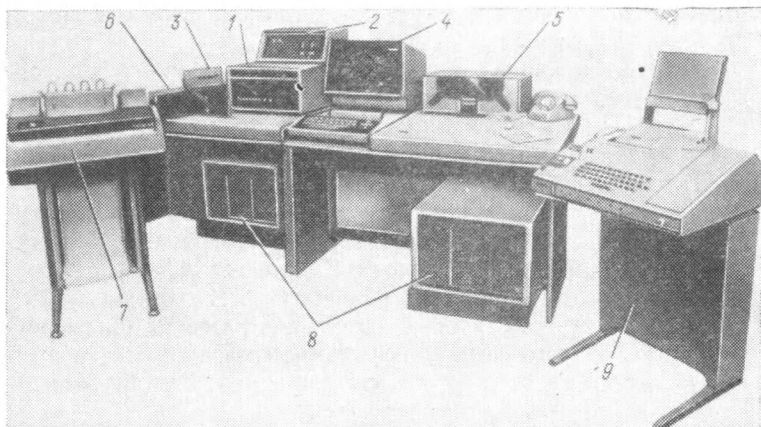


Рис. 19.5.

рассчитаны на работу с гибкими дисками, другие для подключения гибкого диска требуют внесения изменений, которые могут оказаться не очень простыми! Во всяком случае того, кто привык работать с накопителем на гибком диске, трудно заставить использовать какой-нибудь накопитель, имеющий меньшие возможности.

На рис. 19.5 показан набор периферийных устройств, входящих в состав типовой системы разработки (представленные на этой фотографии устройства более подробно описаны в гл. 20): 1 — система разработки; 2 — программатор ППЗУ; 3 — адаптер; 4 — дисплей; 5 — устройство чтения с перфоленты (перфоратор); 6 — микро-ЭВМ; 7 — печатающее устройство; 8 — гибкий диск; 9 — телетайп.

На этом рисунке имеются две микро-ЭВМ. Слева показана микро-ЭВМ, для которой программа разрабатывается.

Из нее временно удален МП. Функции этого МП взяла на себя микро-ЭВМ, входящая в состав системы разработки, подключенной к первой микро-ЭВМ через адаптер. Все периферийные устройства подключены к микро-ЭВМ, входящей в состав системы разработки. Это сделано для облегчения операций тестирования системы.

19.10. Микромашинный тренажер

Помимо больших систем разработки многие изготовители микро-ЭВМ поставляют также микромашинные тренажеры (МТ). Хотя подобные установки иногда называют недорогими системами разработки, они не предназначены для выполнения полного цикла работ по созданию программ от начала до конца. Во многих случаях их память недостаточна для загрузки не только ассемблера, но редактора и имитатора (если только после расширения памяти система не будет дополнена, по меньшей мере, телетайпом или видеотерминалом).

В большинстве МТ периферийное оборудование состоит из шестнадцатиричной клавиатуры (ввод) и дисплея на основе 7-сегментных индикаторов (вывод). Емкость памяти составляет от $1/4$ К до 1 К ячеек ОЗУ и от 1 К до 4 К ячеек ПЗУ.

Программы должны вводиться в шестнадцатиричных кодах, и поэтому функции ассемблера должен выполнять программист. Хотя, как уже было отмечено, МТ создают меньше удобств при разработке программ, они исключительно полезны для изучения возможностей микро-ЭВМ и техники программирования. В табл. 19.1 приведен перечень наиболее известных в настоящее время МТ.

Выводы

1. Диалоговый терминал представляет собой телетайп или клавиатуру и видеотерминалы.
2. Диалоговый терминал используется только в процессе создания рабочей программы.
3. В состав диалогового терминала могут быть включены устройство ввода с перфоленты и ленточный перфоратор.
4. Для хранения больших объемов программ и (или) данных можно использовать накопители на магнитной ленте или гибких дисках.

Т а б л и ц а 19.1 Обзор микромашинных тренажеров

Марка/тип	Тип микропроцессора	Имеющиеся периферийные устройства	Включенные в состав микро-ЭВМ	Питание	Емкость * ОЗУ, Кбайт	Емкость * ПЗУ, Кбайт	Примечания
NEC/TK 80	8080	Шестнадцатирядная клавиатура и дисплей	Нет	Внешнее 12 В (150 мА), 5 В (1 А), земля	1/2 (до 1)	3/4 (до 1)	—
Siemens/Microset 8080	8080	То же	Да	Встроенное	1	1	Интерфейс для подключения кассетного НМЛ
Mostec/KIM1	6502	То же	Нет	Внешнее 5 В (1,2 А), 12 В (100 мА), земля	1	2	То же
Motorola/MEK 6800 D2	6800	То же	Нет	Внешнее +12 В (100 мА), -5 В (1 А), земля	1/4 (до 1/2)	2 (до 4)	То же
Intel/SDK-80	8080	Нет	Нет	То же	1/4 (до 1)	2 (до 4)	Набор конструкторов и комплектующих изделий

Продолжение табл. 19.1

Марка/тип	Тип микро-процессора	Имеющиеся периферийные устройства	Включенные в состав микро-ЭВМ	Питание	Емкость * ОЗУ, Кбайт	Емкость * ПЗУ, Кбайт	Примечания
Intel/SDK-85	8085	Шестнадцатирядная клавиатура	Нет	Внешнее 5 В (1 А)	1/4 (до 1/2)	2 (до 4)	Набор конструктивов и комплектующих изделий, интерфейс для подключения телеаппа
Intel/PROMPT-80	8080	Шестнадцатирядная клавиатура и дисплей	Да	Встроенное	1	4	Модуль программатора ПЗУ (РППЗУ)
AMI/AMI-COS	6800	Тумблеры и светодиоды	Да	Встроенное	1/4 (модули памяти до 64)	1 (модули памяти до 64)	Набор модулей и комплектующих изделий
National/ISP-8P	SC/MP	Шестнадцатирядная клавиатура и дисплей	Да	Встроенное	1/2 (до 2 на каждый дополнительный модуль ОЗУ)	2 (до 4 на каждый дополнительный модуль ОЗУ/ППЗУ)	Интерфейс для подключения телеаппа

* В скобках указаны значения, до которых расширяется память.

Глава двадцатая

ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА

20.1. Введение

Чтобы микро-ЭВМ имела возможность обрабатывать данные, необходимо помимо нее самой иметь также периферийное оборудование. Периферийные устройства могут быть разделены на следующие группы (рис. 20.1): устройства ввода; устройства вывода; устройства управления; внешние запоминающие устройства.

На рис. 20.2 приведена более подробная схема классификации периферийных устройств, которые обычно используются в микромашинных вычислительных системах.

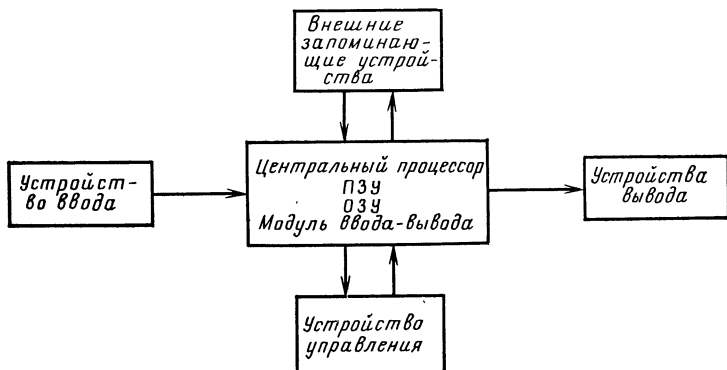


Рис. 20.1.

Многие из устройств, показанных на рис. 20.2, могут быть объединены в одно устройство. Например, в состав телетайпа входят клавиатура, печатающее устройство, устройство ввода с перфоленты, ленточный перфоратор; дисплей включает в себя телевизионный монитор и клавиатуру. Часто телетайп и дисплей используются не только как устройства ввода-вывода, но и как средство управления микро-ЭВМ. Поэтому они описываются в § 20.4, посвященном устройствам управления.

20.2. Устройство ввода

Устройство ввода преобразует вводимые данные из внешней формы представления в электрические сигналы, которые могут восприниматься ЭВМ.

В данном параграфе рассмотрены устройство ввода с перфоленты и устройство ввода с перфокарт.

В устройства, перечисленные на рис. 20.2, включено также и различное оборудование, преобразующее информацию об управляемом процессе в форму, которая воспринимается ЭВМ. Примерами таких устройств являются АЦП, датчики и схемы, используемые для измерения

времени (см. гл. 15). Эти средства в данном параграфе не рассматриваются. Клавишное устройство, с помощью которого вводятся буквы, цифры и прочие символы, рассмотрено в § 20.4.

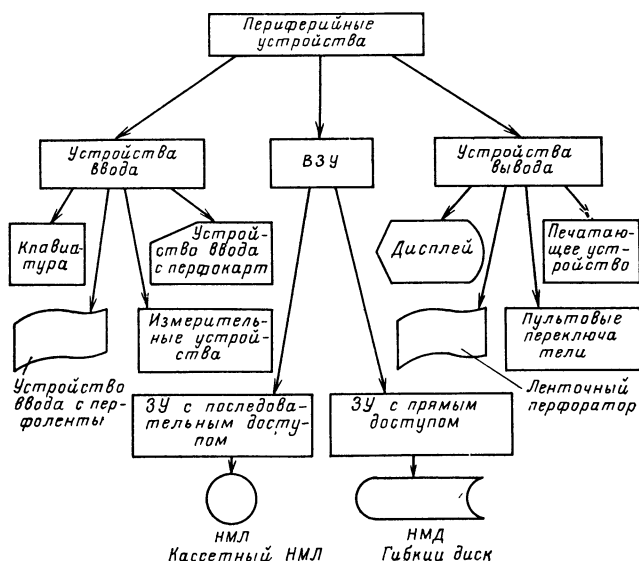


Рис. 20.2.

Устройство ввода с перфоленты. В настоящее время перфолента является самым распространенным носителем информации в микро-ЭВМ. Перфолента разделена на некоторое число дорожек (каналов). На рис. 20.3 показаны наиболее распространенные 7- и 8-канальная перфоленты. При чтении данных перфолента передвигается на одну позицию с помощью синхронизирующих пробивок. На рис. 20.4 показано, каким образом зубчатое колесо осуществляет продвижение перфоленты. Если устройство ввода с перфоленты использует стартовый принцип работы, то для перемещения ленты на одну позицию после чтения очередного символа требуется соответствующий управляющий сигнал.

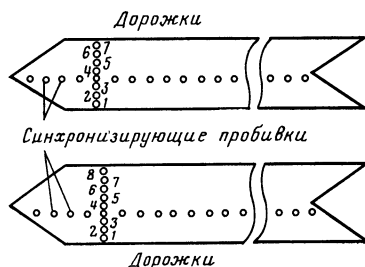


Рис. 20.3.

Существуют устройства ввода с перфоленты с непрерывно движущейся лентой. При каждом проходе синхронизирующей пробивки над читающей головкой формируется стробирующий сигнал, обеспечивающий считывание (чтение) соответствующего символа (читающая головка — это элемент устройства ввода с перфоленты, преобразующий

пробивки на перфоленте в электрические сигналы). Преимуществом устройств ввода с непрерывно движущейся лентой является существенно большая скорость ввода данных, чем у стартстопных устройств. Недостатком является то, что тормозной путь существенно больше расстояния между двумя соседними пробивками.

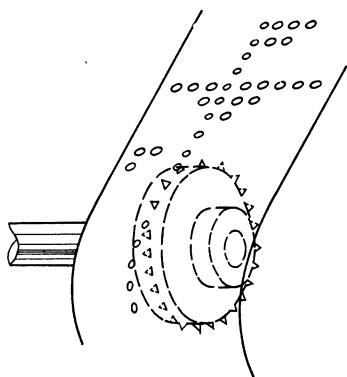


Рис. 20.4.

Чтобы после считывания одной группы символов (блока) не был прочтен ни один символ из следующего блока, между блоками оставляется незаполненное пространство. Оно называется межблочным промежутком. На рис. 20.5 показано, как зубчатые колеса лентопротяжного механизма осуществляют перемещение перфоленты над читающей головкой.

Преобразование пробивок на перфоленте в электрические сигналы осуществляется с помощью фотоэлементов и электроламп. Фотоэлемент облучается световым потоком только при наличии пробивки. При этом генерируется потенциальный сигнал, соответствующий логической единице.

При использовании 7-дорожечной перфоленты имеются восемь фотоэлементов: семь из них служат для обнаружения информационных пробивок и одна для обнаружения синхронизирующей пробивки. С помощью синхронизирующей пробивки распознается нулевой символ.

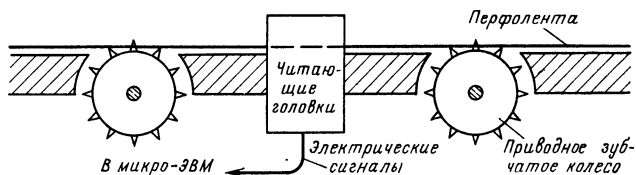


Рис. 20.5.

П р и м е ч а н и е. При использовании 7-дорожечной перфоленты символы могут кодироваться только в коде ASCII (7-битный код). Для 8-дорожечной ленты используется расширенный код обмена информацией EBCDIC (8-битный код) или код ASCII с дополнительным разрядом контроля четности.

Устройство ввода с перфокарт. Перфокарта представляет собой носитель информации, наиболее часто встречаемый в повседневной жизни. Перфокарта имеет 80 столбцов и 12 строк (рис. 20.6). Строки 11 и 12 называются зонными, а остальные десять строк — цифровыми. Срезанный угол перфокарты упрощает проверку правильной их установки в считывающее устройство.

[illegible]

Рис. 20.6.

Достоинством перфокарт является то, что при изменении одной или нескольких входных записей достаточно изменить одну или несколько перфокарт.

Устройство, преобразующее информацию, нанесенную на перфокарты в виде пробивок, в электрические сигналы, называется устройством ввода с перфокарт. Как и при использовании перфоленты, в устройстве ввода с перфокарт используются фотоэлементы.

Обычно за один прием вводится один символ, т. е. содержимое одной колонки перфокарты, поэтому считывающий орган имеет в своем составе 12 фотоэлементов

Для повышения надежности работы устройство ввода с перфокарт обычно имеет в своем составе два считывающих органа. После прохода колонки перфокарты над обоими головками чтения происходит сравнение двух прочитанных символов. Для этого результат первой операции чтения заносится в регистр, в котором хранится до выполнения второй операции чтения. Если сравнение свидетельствует о несовпадении результатов считывания, перфокарта сбрасывается в специальный карман для последующего исправления.

Выводы

1. Устройства ввода преобразуют вводимые данные в электрические сигналы, которые могут обрабатываться в ЭВМ.

2. Наиболее распространенными типами устройств ввода, используемых совместно с микропроцессорами, являются:

а) устройство ввода с перфоленты;

б) устройство ввода с перфокарт.

4. В читающем органе устройство ввода с перфоленты преобразует данные, нанесенные на перфоленту с помощью пробивок, в электрические сигналы. Это обычно осуществляется с помощью фотоэлементов и электроламп.

5. На перфокарте информация представлена пробивками. Каждая перфокарта разделена на 80 колонок и 12 строк.

6. Информация на перфокарту наносится с помощью пробивок.

20.3. Устройства вывода

Устройства вывода данных преобразуют электрические сигналы, поступающие на ЭВМ, в неэлектрические или в аналоговые потенциальные сигналы. Информация может выводиться:

а) в форме, доступной для зрительного восприятия человеком, например в виде букв, цифр или других символов;

б) в виде носителя, который может быть введен позднее в ЭВМ, например перфоленты;

в) в виде сигналов, с помощью которых (используя органы управления) можно непосредственно регулировать производственные процессы (примером управляющих органов являются электромагнитные реле и ЦАП).

В данном параграфе рассматриваются только такие устройства вывода, как перфоратор и печатающее устройство. Видеотерминал рассмотрен в § 20.4.

Ленточный перфоратор осуществляет преобразование электрических сигналов в пробивки на перфоленте. Перфолента в момент формирования пробивок должна быть неподвижной, поэтому все ленточные

перфораторы используют стартстопный принцип работы. Обычно ленточный перфоратор как отдельное устройство встречается редко. Как правило, он входит в состав телетайпа. Это имеет то преимущество, что он может не только работать под управлением ЭВМ, но также использоваться для подготовки перфоленты вручную с помощью клавиатуры. Каждая буква, цифра, или другой символ, который нажимаем на клавиатуре, преобразуется в серию пробивок на перфоленте в коде ASCII.

Печатающее (или *построчно печатающее*) *устройство* представляет собой оборудование, преобразующее информацию, полученную в ЭВМ, в буквы, цифры или иные символы и печатает их на непрерывной бумажной ленте, сложенной в стопку.

На рис. 20.7 приведена фотография печатающего устройства. На выходе печатающего устройства появляется сложенная в стопку бумажная полоса. По краям этой бумажной полосы проделаны отверстия, с помощью которых

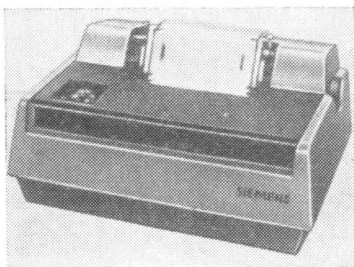


Рис. 20.7.

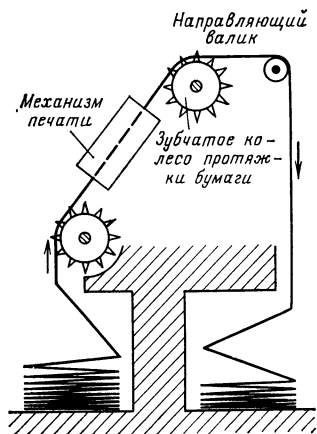


Рис. 20.8.

осуществляется подача бумаги. Через равные промежутки поперек бумажной полосы сделаны просечки. Из рис. 20.8 видно, что стопка бумаги сложена гармошкой до и после печати. Подача бумаги осуществляется с помощью двух зубчатых колес, входящих в зацепление с краевой перфорацией на бумаге. После печати каждой строки зубчатые колеса продвигают бумагу на одну позицию. Набор символов, которые могут печататься на одной строке, зависит от особенностей конкретного печатающего устройства. Стандартными являются строки длиной 32, 64, 80, 120 и 160 символов. Очевидно, что ширина бумаги должна быть приведена в соответствие с числом печатаемых в строке символов.

Очень часто используется многослойная бумага, проложенная копировальной бумагой. Ее применение позволяет получать сразу несколько экземпляров документа.

Выводы

1. Устройства вывода преобразуют выходные данные из машинной формы представления во внешнюю, например в буквы, цифры, пробивки или сигналы, управляющие производственными процессами.

2. Совместно с микро-ЭВМ наиболее часто используются следующие устройства вывода:
 - а) ленточный перфоратор;
 - б) печатающее устройство.
3. Ленточный перфоратор преобразует выводимые данные из машинного представления в пробивки на перфоленте.
4. Печатающее устройство преобразует выводимую из ЭВМ информацию в строки символов, печатаемых на непрерывной бумажной ленте.

20.4. Устройства управления

Через управляющие устройства осуществляется обмен информацией между оператором и ЭВМ. В зависимости от особенностей устройств управления могут выполняться следующие операции:

- а) ввод данных, программ (или частей программ) в основную или внешнюю память;
- б) проверка и изменение содержимого ячеек памяти;

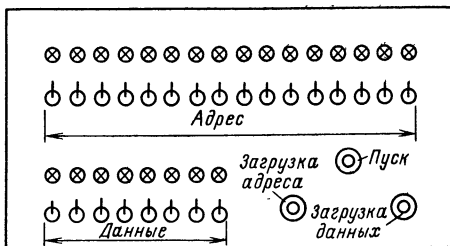


Рис. 20.9.

в) передача управления мониторной программе (например, запуск ассемблера).

В данном параграфе рассматриваются такие устройства управления, как пульт, телетайп и дисплей.

Пульт представляет собой орган управления, с помощью которого, используя тумблеры и световые индикаторы (светодиоды), можно вести диалог с ЭВМ. На рис. 20.9 дан пример пульта. С помощью 16 адресных переключателей можно задать двоичный адрес. Нажатием кнопки ЗАГРУЗКА АДРЕСА можно выбрать соответствующий адрес в памяти. На световом индикаторе данных (восемь ламп) высвечивается содержимое выбранной ячейки памяти. С помощью восьми переключателей можно установить некоторый двоичный код и путем нажатия кнопки ЗАГРУЗКА ДАННЫХ ввести эти данные в соответствующую ячейку памяти.

Для запуска программы используются 16 адресных переключателей, с помощью которых задается адрес первой команды. Если затем нажать кнопку ЗАГРУЗКА АДРЕСА и после этого кнопку ПУСК, то программа начнет работать. Обычно на пульте имеется также кнопка СБРОС.

При нажатии кнопки СБРОС в счетчик команд засылается адрес 0000₁₆. Если кнопка СБРОС отсутствует, то следует установить все переключатели в 0, а затем нажать кнопку ЗАГРУЗКА АДРЕСА.

Телетайп, который был первоначально разработан для телеграфных сообщений, очень похож на пишущую машинку. Он имеет клавиатуру и печатающий механизм, а также устройство ввода с перфоленты и ленточный перфоратор. В телетайпе имеется рулон бумаги в отличие от пишущей машинки, в которую бумага вставляется отдельными листами. При нажатии какой-нибудь клавиши на бумажном рулоне печатается соответствующий символ.

На рис. 20.10 дан внешний вид устройства, сочетающего в себе телетайп с устройством ввода с перфоленты и ленточным перфоратором.

П р и м е ч а н и е. Телетайп печатает символы только прописными буквами и не имеет строчных букв.

На рис. 20.11 приведена диаграмма, иллюстрирующая характер сигналов, посылаемых современным телетайпом в ЭВМ. Каждый символ начинается со стартового бита. Это нулевой сигнал (линия обесточена). Затем следует 8-й бит символа в коде ASCII (в коде ASCII используется всего 7 бит, но обычно добавляется еще один разряд контроля по четности). Десятый бит является стоповым. Стоповый бит всегда равен 1 (линия находится под током).

Старт-бит имеет такую же длину, как и информационные биты. В зависимости от конструкции телетайпа стоп-бит может иметь такую же длину или в 1,5—2 раза большую длины информационных бит. Биты передаются последовательно (последовательный ввод-вывод). Отсюда следует, что между микро-ЭВМ

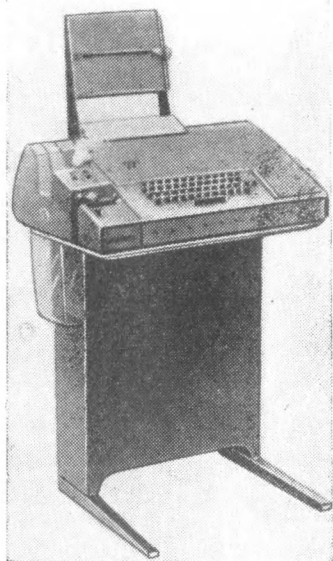


Рис. 20.10.

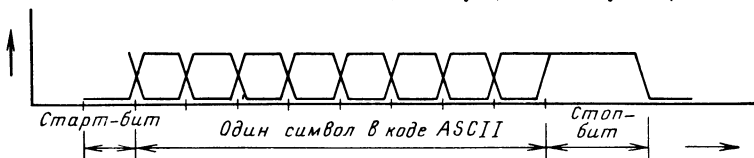


Рис. 20.11.

и телетайпом должна быть предусмотрена интерфейсная схема, выполняющая следующие функции:

- а) согласование скорости работы телетайпа и микро-ЭВМ;
- б) преобразование сигналов от телетайпа в параллельный код при вводе информации; преобразование параллельного кода, поступающего из микро-ЭВМ, в последовательный код телетайпа при выводе

информации (можно написать программу, чтобы преобразование выполняла сама микро-ЭВМ);

в) формирование управляющих сигналов для телетайпа (например, перевод строки, возврат каретки).

Число бит, которые телетайп или иное периферийное устройство может передать или принять в секунду, называется *бод*.

Например, скорость передачи 110 бод означает, что максимальная скорость передачи телетайпа 110 бит/с. При этом длительность каждого бита составляет $1/110 \text{ с} = 9,1 \text{ мс}$.

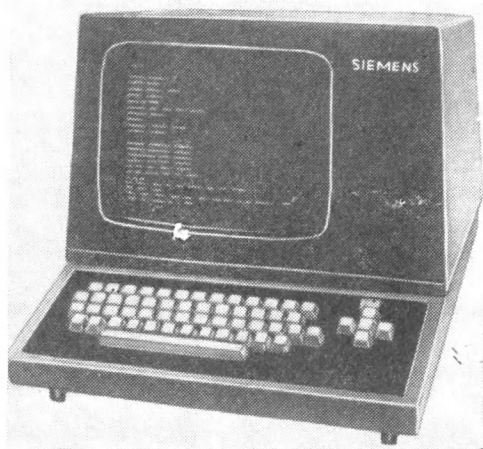


Рис. 20.12.

Если стоп-бит имеет длину, в 2 раза большую длины остальных бит, то длительность передачи одного символа составит $(9 \cdot 9,1) + (2 \cdot 9,1) = 100 \text{ мс}$. Таким образом, в секунду можно передать не более десяти символов.

Дисплей представляет собой телевизионную трубку или монитор, на который информация из ЭВМ выводится в виде слов, чисел или графиков. В большинстве случаев, однако, дисплей связан с клавишным устройством, которое используется как средство ввода информации. Обычно, говоря о дисплее, подразумевают экран и клавиатуру. В данной книге в понятие «дисплей» вкладывается такой же смысл.

Как и телетайп, дисплей может быть использован для выполнения следующих функций:

а) ввода данных;

б) вывода данных;

в) управления работой, т. е., например, изменения содержимого регистров или выдачи команд мониторной программе.

Клавиатура дисплея функционально идентична клавиатуре телетайпа (рис. 20.12). Недостатком использования дисплея является отсутствие «твердой копии» той информации, которая вводится в микро-ЭВМ.

Выводы

1. Пульт — это управляющее устройство, имеющее переключатели и световые индикаторы, с помощью которых можно вести общение с ЭВМ.

2. В состав телетайпа входит клавиатура и печатающее устройство (обычно в состав телетайпа вводят дополнительно устройство ввода с перфоленты и ленточный перфоратор).

3. Число бит в секунду, которое может передать или принять телегайлп, называется бод.

4. Дисплей представляет собой телевизионный экран, на который информация из ЭВМ выводится в виде слов, чисел или графиков.

20.5. Внешние запоминающие устройства

Внешние запоминающие устройства используются для хранения больших объемов данных или программ, которые предполагается использовать впоследствии. Внешние запоминающие устройства принято подразделять на ЗУ с последовательным доступом и ЗУ с прямым доступом. При выборке информации из ЗУ с последовательным доступом приходится просматривать все символы от начала до необходимого места. В ЗУ с прямым доступом ситуация иная: либо нужное место в па-



Рис. 20.13.

мяти непосредственно доступно, либо для доступа требуется несколько операций просмотра. В данном параграфе рассматриваются следующие типы ВЗУ:

- а) накопитель на магнитной ленте (НМЛ);
- б) кассетный накопитель на магнитной ленте (КНМЛ);
- в) накопитель на магнитных дисках (НМД);
- г) накопитель на гибком диске (НГД).

Накопитель на магнитной ленте. Магнитная лента представляет собой пластмассовую основу (обычно типа майлар), на которую нанесено магнитное покрытие. Ширина ленты составляет 12,7 мм. Существуют 7- и 9-дорожечные ленты, имеющие соответственно семь и девять информационных дорожек, расположенных рядом друг с другом. При работе с микро-ЭВМ обычно используется 9-дорожечная лента, на которой информация представлена в коде ASCII.

Биты расположены не по порядку номеров, а таким образом, что менее значимые биты располагаются ближе к кромке ленты (рис. 20.13). Это объясняется тем, что лента чаще всего повреждается по краям. С каждым символом связан бит синхронизации (или тактовый бит), который имеет то же назначение, что и синхронизирующее отверстие на перфоленте. Если в данной позиции записан символ, то бит синхронизации равен 1. Поскольку магнитную ленту нельзя остановить в проме-

жутке между двумя символами данных, данные записываются на ленту и считываются с нее блоками. Между соседними блоками оставляется пустое пространство, называемое межблочным промежутком. Структура расположения информации на ленте показана на рис. 20.14.

Межблочный промежуток имеет стандартную длину 15 мм. Длина блоков может изменяться в определенных пределах. Например, предельными значениями могут быть: не менее 10 и не более 2500 символов в блоке. В начале и конце магнитной ленты записываются специаль-

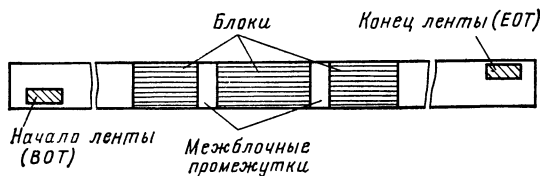


Рис. 20.14.

ные маркеры, обозначающие начало и конец. Они имеют стандартное обозначение ВОТ (начало ленты) и ЕОТ (конец ленты).

Лентопротяжный механизм НМЛ похож на механизм обычного магнитофона. Однако к механизмам запуска и останова предъявляются более жесткие требования, поскольку скорости в НМЛ существенно выше, чем в обычном магнитофоне, а время разгона и торможения существенно меньше. Механизм чтения и записи выполнен подобно тому, как это сделано в обычном магнитофоне (рис. 20.15). Информация с маг-

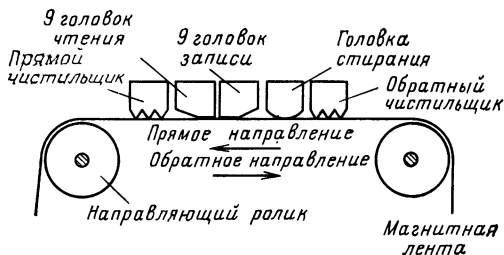


Рис. 20.15.

нитной ленты может считываться при движении ее в прямом и обратном направлениях.

Поскольку имеется всего одна стирающая головка, записывать информацию на ленту можно только при ее движении в прямом направлении. Перед записью всегда следует выполнить стирание. В устройстве предусмотрены средства очистки ленты от пыли при движении в прямом и обратном направлениях. Общий вид НМЛ показан на рис. 20.16.

В зависимости от необходимого объема памяти к ЭВМ может быть подключено несколько стоек с НМЛ. Как и в магнитофонах, кассеты с магнитной лентой можно менять. Следовательно, емкость ленточной памяти является практически неограниченной.

На рис. 20.17 показана катушка с магнитной лентой. Она имеет размеры: длина ленты 730 м, диаметр катушки 26,7 см.

Накопитель на магнитной ленте представляет собой ЗУ с последовательным доступом. Это означает, что для доступа к нужным данным следует просмотреть всю ленту с начала до того места, где находятся данные. Поэтому время доступа к данным на магнитной ленте может быть очень большим (например, около 3 мин). К тому же в НМЛ в малом объеме хранится очень большое количество данных.

Далее перечислены основные параметры, которые характеризуют различные НМЛ.

1. Скорость ленты. Это скорость перемещения ленты по отношению к головкам чтения-записи (стандартными являются следующие значения скорости: 0,95; 1,90; 2,85 и 3,81 м/с).

2. Плотность записи информации. Эта характеристика показывает, какое количество символов (байт) может быть записано на отрезке единичной длины. Обычно плотность записи задается в байтах на миллиметр (стандартными значениями плотности записи являются: 8, 22, 32, 64 и 126 байт/мм).

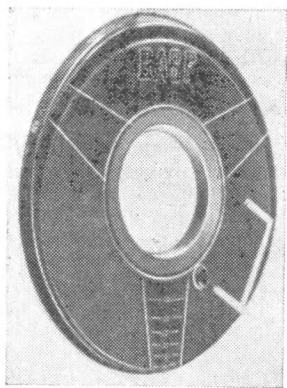


Рис. 20.17.

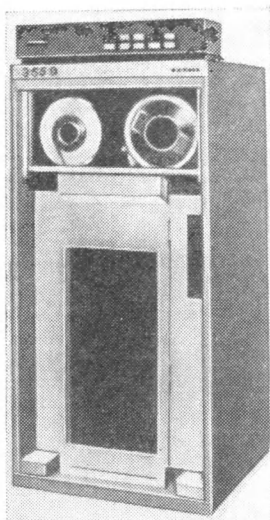


Рис. 20.16.

Пример 20.1. Дано. Накопитель на магнитной ленте имеет скорость 3,81 м/с и плотность записи 64 байт/мм. Требуется определить, какова скорость чтения МЛ.

Решение. Скорость чтения — это максимальное число читаемых символов в секунду. Плотность записи составляет 126 байт/мм, а скорость движения ленты 3810 мм/с. Следовательно, за секунду можно прочесть $64 \times 3810 = 240$ тыс. символов.

Пример 20.2. Дано. Скорость ленты равна 3,81 м/с. Лента имеет длину 720 м.

Требуется. Определить, каково среднее время доступа.

Решение. Чтобы добраться до данных, находящихся в конце ленты, потребуется $720/3,81 = 190$ с.

Поскольку данные в начале ленты доступны непосредственно без просмотра, среднее время поиска составит 95 с, т. е. примерно 1,5 мин.

Кассетный накопитель на магнитной ленте очень похож на обычный кассетный магнитофон. Лента заключена в кассету, которую очень

легко сменить. С помощью интерфейса, функции которого выполняет схема МОДЕМ (МОдулятор — ДЕМОдулятор), обычный кассетный магнитофон может быть подключен к микро-ЭВМ. Недостатком такой схемы является то, что магнитофон не может быть включен и выключен с помощью микро-ЭВМ, и поэтому вся лента должна быть прочитана сразу.

В КНМЛ символы записываются не в параллельном коде, а в последовательном. Плотность записи и скорость обмена информацией у КНМЛ значительно меньше, чем у НМЛ. Однако кассетные накопители значительно дешевле.

Накопители на магнитных дисках. Магнитный диск представляет собой пластмассовую пластинку, покрытую тонким слоем магнитного материала. Пластика (диск) имеет диаметр около 35 см. Диск очень часто сравнивают с грампластинкой. Однако имеются два существенных отличия:

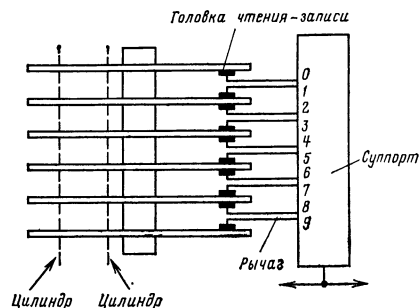


Рис. 20.18.

а) грампластинку можно только воспроизводить (читать), в то время как на диск можно записывать информацию;

2) на грампластинке имеется единственная спиральная бороздка (дорожка), идущая от края к центру диска. На диске имеется 203 концентрические дорожки.

Данные записываются на диск последовательно, как и на кассетный накопитель. Дорожки диска также разделены на блоки. Каждый блок снабжен адресом. Поскольку диск постоянно вращается, не требуется выделения межблочных промежутков. Частота вращения составляет 2400 об/мин. Несколько дисков обычно объединены в пакет, в котором в зависимости от числа пластинок может иметься 2, 10 или 20 рабочих поверхностей. На рис. 20.18 показан пакет, в котором число рабочих поверхностей равно 10. В результате для работы с таким пакетом требуется десять головок чтения-записи. Головки жестко закреплены на рычагах, которые, в свою очередь, закреплены на суппорте. Перемещение суппорта позволяет расположить головку над нужной дорожкой. Все дорожки с одинаковыми номерами образуют цилиндр. Таким образом, в пакете дисков имеются 203 цилиндра, которые имеют номера с 000 по 202. Из 203 цилиндров нормально используются только 200, оставшиеся три являются запасными.

Чтобы выбрать некоторый блок, следует знать:

- номер цилиндра (из диапазона 000—199);
- номер головки (на каждую рабочую поверхность имеется своя головка чтения-записи);
- номер сектора (каждая дорожка разделена на секторы, называемые также блоками).

Суммарное время доступа складывается из следующих времен:

- позиционирования головки на нужный цилиндр (примерно 60 мс);
- выборки нужной дорожки в пределах цилиндра [выборка осуществляется электронной схемой выборки адреса, и время этой операции пренебрежимо мало по сравнению с временем позиционирования головки (см. п. 1)];

3) времени, которое истечет прежде, чем нужный блок пройдет под читающей головкой (так как частота вращения диска равна 2400 об/мин, один оборот совершается за 25 мс. Следовательно, среднее значение задержки равно 12,5 мс).

Таким образом, суммарное время доступа к информации на диске сравнительно невелико. Поэтому НМД относятся к категории ЗУ с прямым доступом. На рис. 20.19 показан пакет дисков в пластиковом защитном контейнере. Такой пакет дисков легко заменяется и емкость памяти на дисках доводится до нужного объема.

На каждой дорожке диска записывается около 32 000 бит, т. е. около 4000 байт. Из этого числа байт 375 байт используется для адресации блоков. Отсюда следует, что емкость дорожки равна 3625 байт. В рассматриваемом НМД имеется 200 цилиндров, в каждом из которых находится десять дорожек. В результате общая емкость пакета равна $200 \cdot 10 \cdot 3625 = 7,25$ млн. байт. Пакет дисков с 20 рабочими поверхностями имеет емкость 14,5 млн. байт.

Накопитель на гибком диске. Гибкий диск представляет собой гибкую пластинку из пластмассы, покрытую магнитным материалом. Он имеет всего одну рабочую поверхность, которая как и в обычном НМД, разделена на концентрические дорожки. Чтение и запись выполняется, как в обычном НМД. Поскольку в гибком диске имеется всего одна рабочая поверхность, нужна единственная головка чтения-записи. Чтобы указать некоторый конкретный блок, достаточно задать номер цилиндра и номер блока (сектора).

Приведенные в данной главе технические характеристики позволяют представить себе порядок величин емкости ЗУ и скорости выборки. В принципе возможны и другие значения рассмотренных параметров.

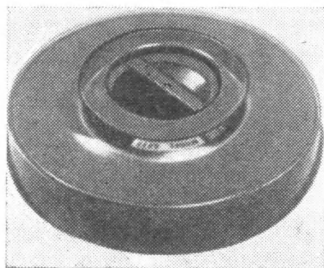


Рис. 20.19.

Выводы

1. Во внешней памяти хранятся большие объемы данных, которые будут далее обрабатываться.

2. Наиболее часто вместе с микро-ЭВМ используются следующие типы ВЗУ:

- а) накопитель на магнитной ленте;
- б) кассетный накопитель на магнитной ленте;
- в) накопитель на магнитном диске;
- г) накопитель на гибком диске.

3. При работе с НМЛ информация вводится блоками. Блоки разделены межблочными промежутками, которые обеспечивают возможность останова ленты и ее разгона до полной скорости.

4. Кассетный НМЛ похож на обычный кассетный магнитофон. Лента заключена в легко сменяемую кассету. Способ записи на ленту последовательный.

5. В НМД данные записываются на пластмассовые диски. Соответствующие дорожки на дисках образуют цилиндры. Дорожки разделены

на блоки, каждый из которых имеет адрес. Поскольку диск непрерывно вращается с постоянной частотой, отпадает необходимость в межблочных промежутках.

6. В отличие от НМЛ накопитель на гибком диске имеет всего один диск. Этот диск выполнен из гибкого материала.

Глава двадцать первая

ИНТЕРФЕЙС ВВОДА-ВЫВОДА МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ

21.1. Введение

Любая прикладная программа должна пройти этапы редактирования, ассемблирования и отладки, в результате чего она будет преобразована в рабочую программу и сможет выполняться в микро-ЭВМ. На каждом из перечисленных этапов программист должен иметь возможность общения с машиной. Так, во время выполнения программы часто оказывается необходимым вводить в машину какие-то дополнительные данные, а также выводить из нее информацию. Подобные операции реализуются с помощью соответствующего периферийного оборудования.

Так как каждое периферийное устройство характеризуется своим особым набором действий, совершаемых при вводе-выводе, средства интерфейса, т. е. сопряжения между микро-ЭВМ и периферией, в каждом случае также должны быть различными. Это означает, что программа, в соответствии с которой происходит обмен данными, строится специальным образом для каждого типа периферийных устройств.

Возможные варианты обмена между микро-ЭВМ и периферийными устройствами можно разделить на три категории.

1. Обмен данными, проходящий в процессе программируемого ввода-вывода. В этом случае момент времени, в который должен начаться обмен данными, определяется ходом выполнения рабочей программы. Конкретный вид команды ввода-вывода, появившейся в рабочей программе, обуславливает организацию нужной последовательности событий и управления в системе.

2. Обмен данными, порождаемый появлением запроса прерывания от УВВ. В этом случае время начала обмена данными определяется работой периферийных устройств.

При появлении запроса прерывания микро-ЭВМ временно прекращает выполнение текущей программы и вводит в действие подпрограмму, специально предназначенную для управления обменом данными.

3. Обмен данными в режиме прямого доступа к памяти (ПДП). Этот режим используется при необходимости передать информацию из внешней памяти в основную. В подобном случае ЦП отключает себя, как это было описано ранее, от шин, вследствие чего внешняя память получает возможность прямого доступа к основной памяти.

Реализация процедур обмена данными требует синхронизации работы ЦП и периферийного устройства. Другими словами, скорости их работы должны согласовываться

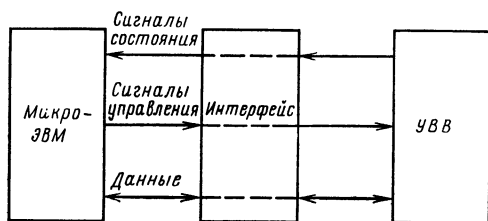


Рис. 21.1.

в том смысле, что при запаздывании во время выполнения операций «медленным» устройством более «быстрое» переходит в состояние ожидания.

Информация, которая передается между устройствами ввода-вывода в микро-ЭВМ, представлена следующими сигналами (рис. 21.1):

1) сигналами состояния, которые сообщают микро-ЭВМ о состоянии УВВ (подано напряжение питания, данные для системы подготовлены и т. д.);

2) сигналами управления, которые представляют собой команды, посылаемые микро-ЭВМ устройствам ввода-вывода;

3) сигналами данных, с помощью которых происходит передача содержательной информации между микро-ЭВМ и УВВ.

Первые две категории сигналов обеспечивают выбор требуемых действий при управлении УВВ. Рассмотрим, например, устройство чтения с перфоленты. При вводе информации в микро-ЭВМ это устройство передает параллельный код считанных им данных на один из портов ввода.

При этом устройство чтения посылает на другой порт по одной из линий соответствующей шины сигнал состояния, указывающий на готовность данных. Тогда по одной из линий шины управления микро-ЭВМ посылает в один из портов вывода команду чтения для данного УВВ. Такой метод взаимодействия микро-ЭВМ с периферийным устройством называют *квтированием* (handshaking).

Рассмотрим процесс обмена данными с применением квтирования несколько подробнее (рис. 21.2). В началь-

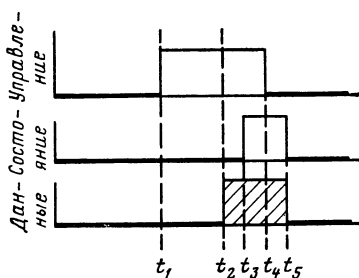


Рис. 21.2.

ответствующих линиях шины управления отсутствуют. Так как каждое действие одного из участвующих в обмене устройств должно вызывать ответную реакцию со стороны другого устройства, необходимо, чтобы эти устройства информировали друг друга о своих состояниях. Если в выполняемой программе в некоторый момент времени появляется команда вво-

да данных, микро-ЭВМ выдает на шину управления соответствующий сигнал (команду) (момент времени t_1 на рис. 21.2). Устройство чтения с перфоленты, получив эту команду, приступает к считыванию данных. Как только процесс считывания завершится (момент времени t_2), полученные с перфоленты данные посылаются на один из портов ввода, после чего устройство чтения выдает сигнал состояния, указывающий на готовность данных для системы (момент времени t_3).

В то время, как устройство чтения выполняет описанные действия, микро-ЭВМ производит периодические проверки, в каком состоянии это устройство находится. Такие проверки продолжают до тех пор, пока состояние устройства не изменится.

Если состояние устройства изменилось, о чем микро-ЭВМ «узнает» во время одной из очередных проверок, она выполняет действия по приему данных, предписываемые командой ввода. После этого микро-ЭВМ снимает сигнал управления с устройства чтения с перфоленты, указывая тем самым на то, что прием данных завершен (момент вре-

мени t_4). В ответ на это устройство чтения снимает сигнал состояния готовности данных (момент времени t_5).

Такой метод работы позволяет организовать управление периферийными устройствами по командам, содержащимся в рабочей программе.

21.2. Программируемый ввод-вывод

В том случае, когда необходимость обмена данными возникает в ходе выполнения рабочей программы, задача организации управления вводом-выводом в микро-ЭВМ целиком ложится на программиста.

Для ввода или вывода используются двухбайтные команды IN или OUT.

Первый байт содержит код операции, указывающий на необходимость ввода или вывода данных, в то время как во втором байте содержится указание на источник или место назначения передаваемой информации, т. е. во втором байте содержится номер порта УВВ.

При программируемом вводе-выводе могут возникнуть трудности из-за разницы в быстродействии микро-ЭВМ и периферийного оборудования. Например, при выполнении операций вывода данных может случиться, что микро-ЭВМ уже начала выдачу данных, а в это время устройство вывода еще занято обработкой предыдущей порции информации. Во время операции ввода может возникнуть ситуация, когда микро-ЭВМ несколько раз подряд считывает одни и те же данные из-за того, что периферийное устройство еще не успело подготовить следующую порцию данных.

Вывод

1. Обмен данными между периферийными устройствами и микро-ЭВМ может произойти в процессе программирования ввода-вывода или в результате появления запроса прерывания от УВВ.

2. При программируемом вводе-выводе могут возникнуть трудности из-за разницы в быстродействии микро-ЭВМ и периферийных устройств.

21.3. Программируемый ввод с квитированием

Надежная синхронизация работы устройств ввода и микро-ЭВМ при выполнении операции ввода может быть достигнута с помощью тестовых проверок.

Для этой цели устройство ввода посылает в микро-ЭВМ сигнал, указывающий на готовность новой порции данных. Сигнал «данные для ввода подготовлены» обозначим DAVIN (data in available). Перед началом выполнения операции ввода микро-ЭВМ должна проверить, имеется ли на ее входе такой сигнал.

В свою очередь, микро-ЭВМ должна указать, окончила ли она обработку предыдущей порции информации и готова ли к приему следующей, так как некоторые устройства ввода выдают информацию с весьма высокой частотой и может оказаться, что микро-ЭВМ, все еще обрабатывая предыдущую порцию данных, теряет следующую порцию, передаваемую устройством

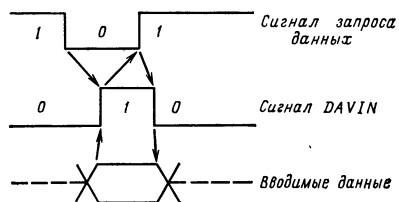


Рис. 21.3.

При обмене информацией с большинством периферийных устройств вопросы и ответы даются в следующей последовательности.

Генерируя так называемый сигнал запроса данных Q, микро-ЭВМ

тем самым указывает на готовность к приему следующей порции данных. Следовательно, прежде чем начать передачу этих данных, периферийное устройство сначала должно проверить, чему равен сигнал запроса данных — 0 или 1. Обмен управляющими сигналами (DAVIN и сигнал запроса Q) лежит в основе метода передачи с квитированием.

На рис. 21.3 приведена временная диаграмма описанного процесса тестовой проверки. Стрелками обозначена взаимосвязь сигналов друг с другом.

Если сигнал запроса данных становится равным 0, то это воспринимается устройством ввода как указание на то, что микро-ЭВМ закончила обработку предыдущей порции данных и готова к приему следующей. В этом случае устройство ввода передает новые данные микро-ЭВМ и одновременно устанавливает сигнал DAVIN в 1. Получив этот сигнал, микро-ЭВМ принимает данные, после чего снимает сигнал запроса данных (устанавливает его в 1). Это означает, что микро-ЭВМ закончила прием данных и занята их обработкой. Так как порция данных передана в микро-ЭВМ, то для устройства ввода отпадает необходимость ее

дальнейшего сохранения. Когда в микро-ЭВМ обработка данных заканчивается, она вновь устанавливает сигнал запроса данных равным 0, после чего описанная последовательность действий может повториться.

На рис. 21.4 изображена блок-схема процедуры, в соответствии с которой микро-ЭВМ реализует описанный процесс ввода данных с применением квити-рования. Процедура оформляется в виде программы с именем DATA IN. При обращении к этой подпрограмме происходит ввод одного слова. В блоке 1 микро-ЭВМ запрашивает данные для ввода, устанавливая сигнал запроса данных равным 0.

Блоки 2 и 3 образуют цикл, который повторяется до тех пор, пока сигнал DAVIN не станет равным 1. Если сигнал DAVIN принимает значение 1, то это означает, что готова новая порция данных. В блоке 4 эти данные передаются в микро-ЭВМ. В блоке 5 микро-ЭВМ устанавливает сигнал запроса данных в 1, после чего с помощью блоков 6 и 7 организуется ожидание того момента, когда сигнал DAVIN станет равным 0. Когда этот момент наступает, микро-ЭВМ пересыла-

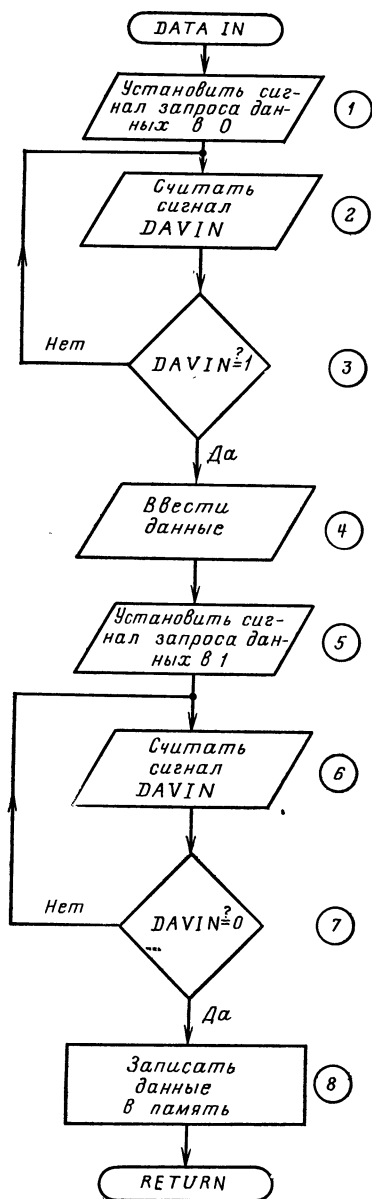


Рис. 21.4.

ет в память данные, которые были введены в блоке 4, и возвращается к выполнению основной программы.

Замечание

В некоторых случаях при выполнении операции ввода проверку можно производить с помощью лишь одного сигнала. Этот сигнал, называемый стробом от периферийного устройства, генерируется устройством ввода и указывает, готовы или нет данные, предназначенные для ввода в микро-ЭВМ. При появлении сигнала «строб от периферийного устройства», что соответствует готовности данных, микро-ЭВМ эти данные должна принять, уложившись в заранее заданный промежуток времени (например, 10 мс). После кратковременного переключения сигнала стробирования в 0 и его возврата в 1 микро-ЭВМ может принять очередную порцию данных.

Вывод

1. Квити́рование используется для надежной синхронизации работы микро-ЭВМ и периферийного оборудования.

2. Если ввод программируется с применением квитирования, то микро-ЭВМ посылает сигнал запроса данных устройству ввода, что указывает на готовность микро-ЭВМ к приему новых данных.

3. Периферийное устройство посылает сигнал DAVIN в микро-ЭВМ, указывая с его помощью, готовы или нет данные, предназначенные для ввода.

21.4. Программируемый вывод с квитированием

При выполнении операции вывода информации устройство вывода должно посылать в микро-ЭВМ сигнал, указывающий на готовность устройства к приему новых данных и называемый сигналом занятости устройства. Перед началом вывода данных микро-ЭВМ должна произвести проверку этого сигнала.

В свою очередь, микро-ЭВМ должна сигнализировать о том, что

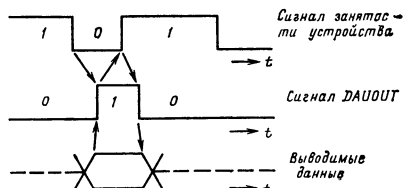


Рис. 21.5.

новая порция данных готова для передачи устройству вывода. Сигнал «данные для вывода подготовлены» обозначим DAVOUT (data out available). Таким образом, прежде, чем начать прием данных, устройство вывода должно проверить, присутствует ли на его входе сигнал DAVOUT.

На рис. 21.5 изображена временная диаграмма процесса тестовой проверки, причем, как и на рис. 21.3, стрелками обозначена взаимосвязь сигналов друг с другом.

В том случае, когда устройство вывода устанавливает нулевое значение сигнала занятости устройства, оно указывает тем самым на свою готовность к приему новых данных. Это позволяет микро-ЭВМ приступить к выводу данных, причем одновременно она выдает сигнал DAVOUT, равный 1. Данные в результате принимаются устройством вывода, что освобождает микро-ЭВМ от необходимости сохранять на выходе значения выводимых данных и сигнала DAVOUT. Как только устройство вывода заканчивает обработку данных, оно устанавливает сигнал занятости устройства

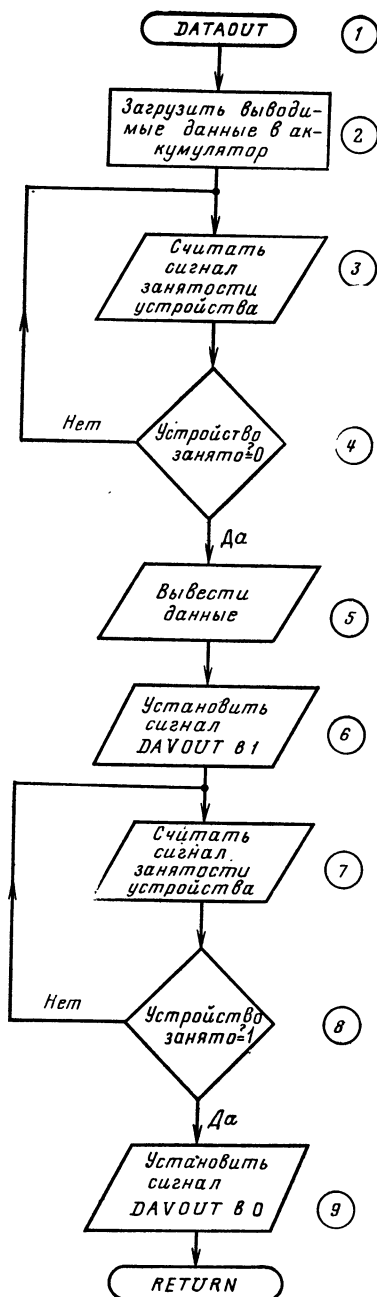


Рис. 21.6.

равным 0, после чего описанная последовательность действий может повторяться.

На рис. 21.6 изображена блок-схема подпрограммы вывода, которая реализует рассмотренную выше процедуру. После передачи управления подпрограмме (блок 1) в блоке 2 производится подготовка данных для вывода, которая заключается в пересылке данных из памяти в аккумулятор. Цикл, образованный блоками 3 и 4, повторяется до тех пор, пока сигнал занятости устройства не примет значение 0. После выполнения этого условия происходит вывод данных в блоке 5 с помощью команды OUT; сигнал DAVOUT устанавливается в 1. Блоки 7 и 8 образуют другой цикл, выполняемый до тех пор, пока устройство вывода не установит сигнал занятости устройства в 1, т. е. пока оно не завершит прием данных. Когда это произойдет, микро-ЭВМ установит сигнал DAVOUT в 0 и вернется к выполнению основной программы.

Выводы

1. При программировании вывода с применением квинтирования устройство вывода посылает в микро-ЭВМ сигнал занятости устройства, который указывает на готовность периферийного устройства к приему новых данных.

2. Микро-ЭВМ посылает в периферийное устройство сигнал DAVOUT, который указывает на готовность данных для вывода из микро-ЭВМ.

21.5. Недостатки программируемого ввода-вывода

В предыдущих параграфах было показано, каким образом использование тестовых сигналов позволяет надежно синхронизировать работу микро-ЭВМ и периферийного устройства. Рассмотрение такого режима работы позволяет заметить, что в процессе обмена данными действия микро-ЭВМ сводятся лишь к операциям ввода-вывода, в промежутке между которыми она ожидает сигналов от внешних устройств, не выполняя никаких полезных действий.

Такое ограничение функций микро-ЭВМ лишь выполнением операций ввода и вывода заметно снижает эффективность работы системы.

Решение проблемы состоит в использовании методов обмена информацией по прерываниям от УВВ, которые рассмотрены в следующем параграфе.

21.6. Ввод-вывод информации по прерыванию

Если микро-ЭВМ обладает средствами обработки запросов прерывания, поступающих с периферийных устройств, выполнение основной программы может чередоваться с выполнением подпрограмм ввода-вывода. Если, например, в устройстве ввода возникает необходимость передать данные в микро-ЭВМ, то оно производит запрос прерывания (рис. 21.7).

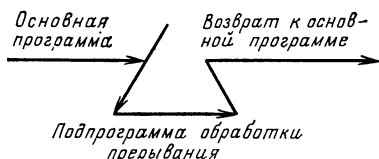


Рис. 21.7.

Это означает, что устройство ввода посылает сигнал на вход «прерывание» микропроцессора. Как только МП обнаруживает сигнал прерывания, он останавливает выполнение основной программы и передает управление подпро-

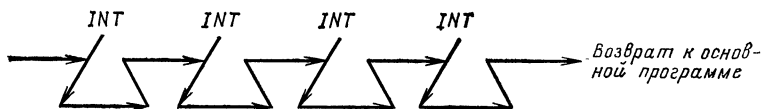


Рис. 21.8.

грамме ввода данных, которая носит название «подпрограмма обработки прерывания». По завершении выполнения подпрограммы обработки прерывания происходит возврат к основной программе. На рис. 21.8 иллюстрируется процесс ввода и вывода некоторой последовательности данных.

21.7. Вывод данных по прерыванию

Если предполагается, что вывод данных будет производиться по сигналу прерывания, то окажется весьма полезным предварить работу программы обработки прерывания подпрограммой подготовки данных к выводу. Назначением последней является размещение данных, предназначенных для вывода, по фиксированным адресам памяти. Эту подпрограмму можно расположить, например, в начале основной программы. Теперь, если устройство вывода посылает запрос прерывания, команду OUT можно использовать непосредственно после выборки данных из памяти.

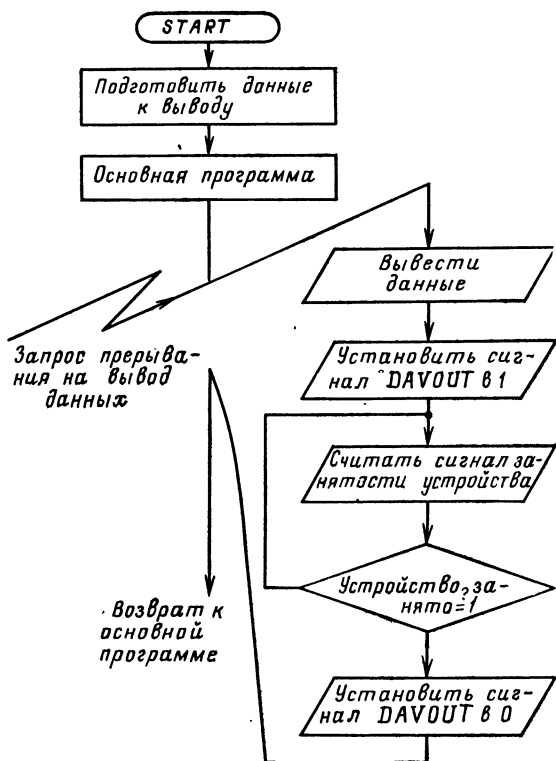


Рис. 21.9.

На рис. 21.9 представлена блок-схема подпрограммы обработки запроса прерывания на вывод данных.

21.8. Ввод данных по прерыванию

На рис. 21.10 представлена блок-схема подпрограммы обработки запроса прерывания на ввод данных. В начале основной программы микро-ЭВМ устанавливает сигнал запроса данных в 0, что позволяет периферийному устройству прервать выполнение основной программы в любой момент времени.

Заметим, что подпрограммы обработки прерывания не обязательно предназначаются лишь для ввода или вывода данных. Они могут использоваться и для других целей.

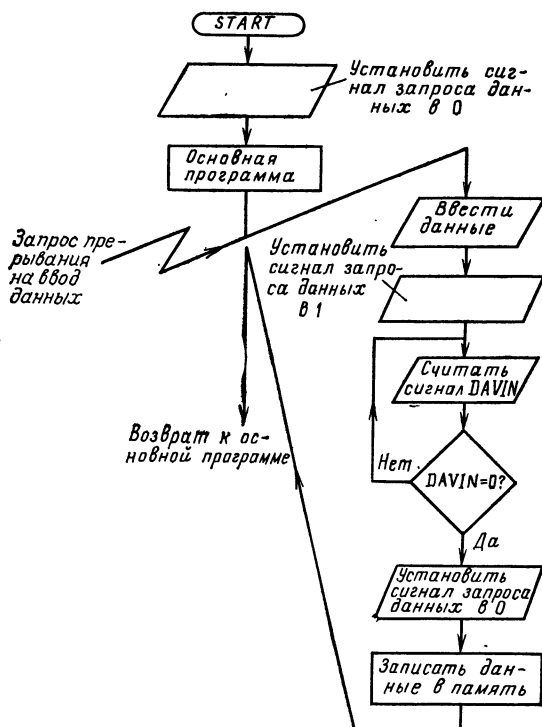


Рис. 21.10.

Выводы

1. Недостатком программируемого ввода-вывода является то, что при обмене данными с периферийными устройствами микро-ЭВМ значительную часть времени находится в состоянии ожидания.

2. При обмене данными по прерыванию выполнение основной программы чередуется с работой подпрограмм ввода-вывода.

3. Подпрограмма обработки прерывания отвечает за организацию обмена данными между микро-ЭВМ и периферийными устройствами при выдаче последним запроса прерывания.

21.9. Вектор прерывания

Микро-ЭВМ обычно имеет набор УВВ, что приводит к необходимости наличия соответствующего числа линий, по которым должны передаваться сигналы запросов прерывания. После появления запроса прерывания происходит передача управления нужной подпрограмме обработки прерывания.

В большинстве МП сигналу запроса прерывания соответствует набор из нескольких (обычно трех) дополнительных бит, которые МП может использовать для формирования адреса соответствующей подпрограммы. В этом случае речь идет о наличии векторной системы прерываний.

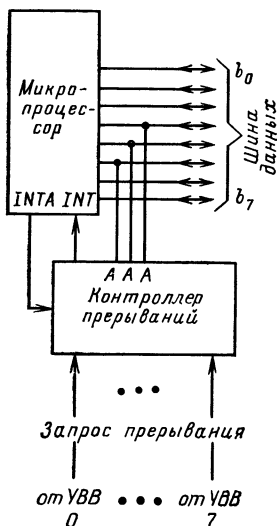


Рис. 21.11.

Содержимое ячейки памяти с этим адресом и двух следующих ячеек образует команду передачи управления, по которой совершается переход к первой команде требуемой подпрограммы обработки прерывания. На рис. 21.11 иллюстрируется организация системы при векторном прерывании.

В определенный момент времени УВВ посылает запрос прерывания через контроллер прерываний на вход INT микропроцессора. При появлении запроса прерывания МП выдает

сигнал подтверждения прерывания на выход $INTA$; этот сигнал передается контроллеру прерываний. В результате контроллер прерываний посылает на шину данных код, который указывает, с какого именно периферийного устройства поступил запрос прерывания. Естественно, каждому УВВ присваивается свой собственный код. Например, для телетайпа это может быть $CF_{(16)} = 11001111_{(2)}$, для устройства перфорации карт $EF_{(16)} = 11101111_{(2)}$ и т. д. В общем виде такой код выглядит следующим образом: $11AAA111$.

В зависимости от того, какое УВВ послало запрос прерывания, контроллер прерываний задает нужную комбинацию

нацию значений символов AAA. Формируемая таким образом команда носит название команды повторного запуска (restart). С помощью этой команды МП прерывает выполнение основной программы и переходит к обслуживанию запроса прерывания.

Выполнение команды повторного запуска порождает следующую последовательность действий.

1. Содержимое счетчика команд запоминается в стеке, а указатель стека устанавливается соответствующим образом. Этим обеспечивается возврат к основной программе по завершении выполнения подпрограммы обработки прерывания.

2. В счетчик команд загружается значение $0000000000AAA\ 000_{(2)}$. Это значение определяет адрес, с которого начинает выполняться подпрограмма обработки запроса прерывания.

При появлении запроса прерывания, например, от телетайпа контроллер прерываний задает символу AAA значение $001_{(2)}$. В этом случае в регистр команды будет загружено

значение $11\ 001\ 111_{(2)}$. Так как $001_{(2)} = 1_{(10)}$, то формируется команда ПОВТОРНЫЙ ЗАПУСК 1. После этого программа выполняется начиная с адреса $0000000000\ 001\ 000_{(2)} = 0008_{(16)}$. Содержимое ячейки памяти с этим адресом и двух следующих ячеек образуют, как уже говорилось, команду передачи управления. Если эта команда, например, имеет вид JMR 2000H, то адресом первой команды подпрограммы обработки прерывания будет $2000_{(16)}$. Предположим, что для другого УВВ контроллер прерываний задает символу AAA значение $000_{(2)}$. Тогда в регистр команды загружается значение $11\ 000\ 111_{(2)}$. В результате образуется команда ПОВТОРНЫЙ ЗАПУСК 0 [так как $000_{(2)} = 0_{(10)}$] и программа продолжает работу начиная с адреса $0000000000\ 000\ 000_{(2)} = 0000_{(16)}$, по которому записан первый байт команды передачи управления.

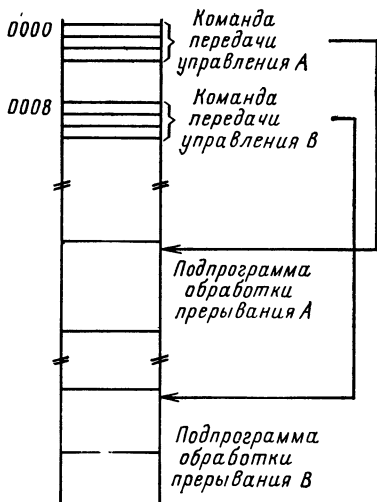


Рис. 21.12.

В силу того, что символы AAA представляют собой комбинацию из 3 бит, может существовать $2^3 = 8$ различных команд повторного запуска.

На рис. 21.12 представлена область памяти микро-ЭВМ, в которой выделены две хранящиеся в ней команды передачи управления и соответствующие каждой из них подпрограммы обработки прерывания. Например, при выполнении команды ПОВТОРНЫЙ ЗАПУСК 1 в счетчик команд загружается значение $0000000000\ 001\ 000_{(2)} = 0008_{(16)}$. Содержимое ячейки памяти с этим адресом и двух следующих ячеек образует команду передачи управления первой команде той подпрограммы обработки прерывания, которая соответствует данной команде повторного запуска. Так же, как и в обычной подпрограмме, последней в подпрограмме обработки прерывания является команда возврата. Этим обеспечивается загрузка в счетчик команд значения, которое было помещено для хранения в стек при выполнении команды повторного запуска.

После завершения выполнения подпрограммы обработки прерывания управление возвращается основной программе в той точке, в которой ее работа была приостановлена из-за появления запроса прерывания.

Замечания

1. Как и в обычных подпрограммах, существует возможность использования вложенных подпрограмм обработки прерывания. В подобном случае выполнение какой-то подпрограммы обработки прерывания приостанавливается запросом прерывания, поступающим от другого УВВ. При этом существует возможность установления системы приоритетов для запросов прерывания от различных УВВ. В этом случае контроллер прерываний должен следить за тем, чтобы последовательность обработки двух одновременно появившихся запросов прерывания соответствовала их приоритетам.

2. Во время выполнения подпрограммы обработки прерывания может измениться содержимое какого-либо из регистров ЦП, например регистра признаков. Поэтому рекомендуется запоминать содержимое регистров МП перед началом выполнения подпрограммы обработки прерывания, что может быть сделано, например, путем пересылки содержимого регистров в стек.

Выводы

1. Вектор прерываний представляет собой адрес памяти, который ЦП формирует по команде повторного запуска.

2. Содержимое ячейки памяти с таким адресом и двух следующих ячеек образует команду передачи управления первой команде запрошенной подпрограммы обработки прерывания.

3. Команда повторного запуска, которая формируется контроллером прерываний, представляет собой особый вид команды CALL; она имеет однобайтный формат и содержит вектор прерывания.

21.10. Прямой доступ к памяти

Если возникает необходимость обмена большими массивами данных между микро-ЭВМ и периферийными устройствами, то это может быть выполнено или с помощью программируемого ввода-вывода или по прерыванию. Однако в обоих случаях значительное количество времени будет потеряно на выполнение вспомогательных команд, не всегда несущих полезную нагрузку. Например, при использовании программируемого ввода-вывода микро-ЭВМ в ожидании данных выполняет специальный цикл, пока не будут выполнены команды ввода-вывода, а при вводе-выводе по прерыванию команды PUSH и POP.

Эти непроизводительные потери можно исключить, введя в состав микро-ЭВМ контроллер прямого доступа к памяти. Контроллер ПДП (рис. 21.13) обеспечивает реализацию квитирования в процессе взаимодействия микро-ЭВМ с периферийными устройствами и, кроме того, определяет адрес ячейки памяти, по которому должна произойти запись или выборка данных. Для этой цели контроллер ПДП подключается непосредственно к шинам микро-ЭВМ, в результате чего периферийные устройства получают возможность прямого доступа к памяти. Режим ПДП реализуется без участия ЦП. Единственным условием, предъявляемым к ЦП, является то, что он должен, как говорилось ранее, отключаться от шин в то время, когда они используются в режиме ПДП. Это достигается с помощью буферных усилителей с тремя состояниями.

Для обмена данными с основной памятью контроллеру ПДП требуется следующая информация.

1. Число слов, передаваемых при обмене данными (число байт в 8-разрядной микро-ЭВМ). Для хранения этого числа в контроллере ПДП имеется счетчик данных.

2. Адрес первого слова памяти, начиная с которого будет производиться запись или считывание данных. Для хранения этого адреса в контроллере ПДП предназначен регистр адреса.

3. Управляющее слово, указывающее вид обмена данными — считывание или запись. Для хранения управляющего слова в контроллере ПДП предусмотрено наличие регистра состояния.

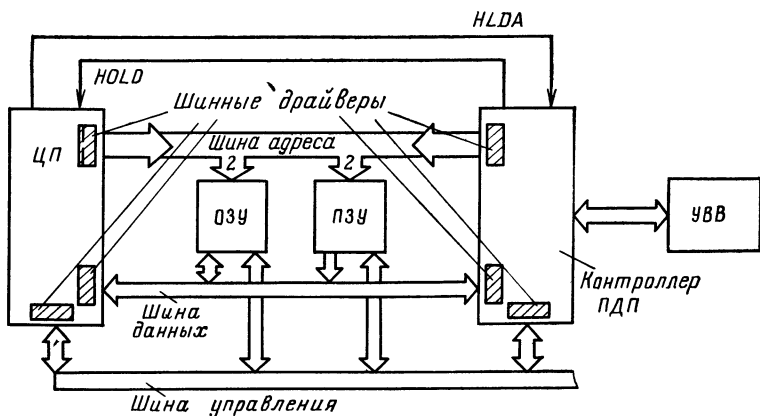


Рис. 21.13.

После передачи центральным процессором всей этой информации контроллеру ПДП контроллер посылает ЦП сигнал захвата шин HOLD. В ответ ЦП переводит буферные усилители в высокоимпедансное состояние и посылает контроллеру ПДП сигнал подтверждения захвата шин HLDA. После этого начинается передача данных, по окончании которой контроллер ПДП снимает сигнал HOLD со входа ЦП и ЦП снова подключается к шинам.

ПРИЛОЖЕНИЕ

СИСТЕМА КОМАНД МИКРОПРОЦЕССОРА 8080

Электронная вычислительная машина (независимо от ее сложности) может делать только то, что ей «говорят». Этот «текст» представляет собой последовательность команд, при-

надлежащих *программе*. К программисту относится область ПО, а разработчику принадлежит область аппаратных средств. К программному обеспечению ЭВМ относятся и те программы, которые написаны для этой ЭВМ.

В процессе проектирования ЭВМ разработчики обеспечивают возможность выполнения некоторого набора операций в ЦП. Центральный процессор спроектирован таким образом, что какая-либо операция выполняется только тогда, когда управляющее устройство дешифрирует код соответствующей команды. Следовательно, операции, которые могут быть выполнены ЦП, определяются *списком (набором) команд*.

Каждая команда предоставляет программисту возможность выполнить в ЭВМ соответствующую операцию. Все ЭВМ имеют в системе команд команды арифметических операций, таких как сложение содержимого двух регистров. Часто в системе команд имеются команды логических операций (например, операция логического ИЛИ над содержимым двух регистров) и команды регистровых операций (например, операция инкрементирования содержимого регистра). Система команд ЭВМ должна также включать команды, реализующие обмен данными между регистрами и памятью и между регистрами и УВВ. Большинство систем команд ЭВМ содержит, кроме того, *условные команды*. Условная команда задает некоторую операцию только в том случае, если реализуется заданное условие, например переход к определенной команде, если результат выполнения предыдущей команды нулевой. Условные команды предоставляют программе возможность принятия решений.

Задавая логически связанную последовательность команд в программе, программист может «диктовать» вычислительной машине выполнение обычных и весьма специфических функций.

Электронная вычислительная машина, однако, может выполнять программы, все команды которой представлены в двоичном коде (наборами 0 и 1), называемом *машинным кодом*. Так как программирование в машинных кодах предельно затруднено, то получили развитие языки программирования. Имеются программы, с помощью которых команды, записанные на языке программирования, переводятся в машинные коды процессора.

Одним из языков программирования является *язык ассемблера*. Специальные мнемокоды языка ассемблера соответствуют каждой команде ЭВМ. Программист может писать

программу (называемую *исходной программой*) с помощью этих мнемочкодов и некоторых операндов. Исходная программа затем переводится в машинные команды (называемые *объектными кодами*). Каждая команда на языке ассемблера преобразуется в одну машинную команду (один или несколько байт) с помощью *ассемблирующей программы*. Язык ассемблера является обычно машиннозависимым языком, т. е. языком, который применим только к одному типу ЭВМ.

Типы команд микропроцессора 8080

Система команд МП 8080 состоит из пяти различных типов (групп) команд.

Группа команд пересылки выполняет операции обмена данными между регистрами или между регистрами и памятью.

Группа команд арифметических операций выполняет операции сложения, вычитания, инкрементирования или декрементирования данных в регистрах или ячейках памяти.

Группа команд логических операций выполняет операции И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ, сравнения, сдвига или дополнения данных в регистрах или ячейках памяти.

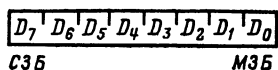
Группа команд ветвления выполняет операции условного и безусловного переходов по программе, условного и безусловного вызовов подпрограммы, а также условного и безусловного возвратов из подпрограммы.

Группа команд управления и операций со стеком и УВВ выполняет операции ввода-вывода, операции со стеком и операции управления признаками состояний.

Форматы команд и данных

Память МП 8080 представляет собой массив 8-битных слов, называемых байтами. Каждый байт имеет свой 16-битный адрес, определяющий его положение в последовательности ячеек памяти. Микропроцессор 8080 может прямо адресовать 65 536 байт памяти, которая может содержать как ПЗУ, так и ОЗУ.

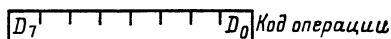
Данные в МП 8080 хранятся в памяти в виде 8-битных слов:



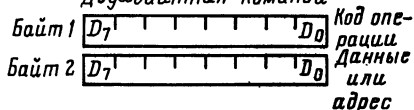
Если содержимое регистра или слово в памяти представляет собой двоичное число, то необходимо индексировать каждый бит этого слова. В МП 8080 *младшим значащим битом* (МЗБ) является бит 0, а бит 7 (с порядковым номером 8) является *старшим значащим битом* (СЗБ).

Команды МП 8080 имеют одно-, двух- или трехбайтный формат. Многобайтные команды должны быть размещены в последовательных ячейках памяти; адрес первого байта команды является адресом всей команды. Формат команды зависит от особенностей выполняемой операции:

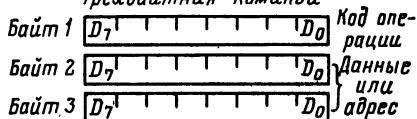
Однobaйтная команда



Двухбайтная команда



Трехбайтная команда



Способы адресации

Часто данные, которые участвуют в операции, хранятся в памяти. Когда используются многобайтные данные, они подобно командам располагаются в последовательных ячейках памяти, в порядке возрастания от младшего значащего байта к старшему. Микропроцессор 8080 имеет четыре раз-

личных способа адресации данных, хранимых в памяти или в регистрах.

Прямая адресация. Вторым и третий байты команды содержат точный адрес байта данных в памяти (младший полуадрес представлен вторым байтом команды, старший полуадрес — третьим).

Регистровая адресация. В коде команды адресуется регистр или пара регистров, в которых хранятся данные.

Косвенно-регистровая адресация. Команда выбирает регистровую пару, в которой содержится адрес ячейки памяти (старший полуадрес располагается в первом регистре пары, а младший — во втором).

Непосредственная адресация. Данные представлены в теле команды. Данные могут быть 8- или 16-битными (первым следует младший значащий байт, вторым — старший).

Если отсутствуют прерывания и команды ветвления, то выборка и исполнение команд осуществляются последовательно из ячеек памяти, адреса которых возрастают (инкрементируются). Команды ветвления могут задавать адрес следующей команды одним из двух способов.

Прямой адресацией. Команда ветвления содержит адрес команды, которая должна выполняться следующей (за исключением команды RST, второй байт команды содержит младший полуадрес, а третий байт — старший полуадрес следующей команды).

Косвенно-регистровой адресацией. Команда ветвления задает регистровую пару, содержимое которой представляет собой адрес следующей команды (старший полуадрес расположен в первом, а младший — во втором регистре пары).

Команда RST является специальной однобайтной командой вызова, используемой обычно для реализации прерываний. Команда RST содержит 3-разрядное поле, в котором содержится один из восьми возможных адресов управляющей программы.

Признаки состояния

Команды МП 8080 оперируют с пятью признаками состояния: НУЛЬ (Zero), ЗНАК (Sign), ПАРИТЕТ (Parity), ПЕРЕНОС (Carry), ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС (Auxiliary Carry), каждый из которых занимает 1 бит реги-

стра признаков в ЦП. Признак имеет место, если данный бит установлен в состояние 1; признак отсутствует, если данный бит установлен в состояние 0.

Если не рассматривать все возможные случаи, при которых команды изменяют содержимое регистра признаков, то признаки формируются следующим образом:

НУЛЬ: если результат выполнения команды нулевой, то признак устанавливается в 1, в противном случае — в 0.

ЗНАК: если старший значащий бит результата операции имеет значение 1, то признак устанавливается в 1, в противном случае в 0.

ПАРИТЕТ: если сумма по модулю 2 всех разрядов результата операции равна нулю (если число единиц в результате четное), то признак устанавливается в 1, в противном случае (если число единиц в результате нечетное) — в 0.

ПЕРЕНОС: если в результате выполнения команды из старшего разряда возникает перенос (при суммировании) или займ (при вычитании или сравнении), то признак устанавливается в 1, в противном случае — в 0.

ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС: если в результате выполнения операции возникает перенос из разряда 3 в разряд 4, то признак устанавливается в 1, в противном случае — в 0. Этот признак возникает при выполнении операций сложения, вычитания, инкрементирования, декрементирования, сравнения и логических операций, но используется только по команде DAA (десятичная коррекция) при выполнении операций сложения и инкрементирования десятичных двоично-кодированных чисел.

Символы и аббревиатуры

Ниже приводятся символы и аббревиатуры, которые используются при описании команд МП 8080.

СИМВОЛ	ЗНАЧЕНИЕ
accumulator	Регистр А
addr	16-битный адрес
data	8-битные данные
data 16	16-битные данные
byte 2	Второй байт команды
byte 3	Третий байт команды
port	8-битный адрес УВВ
r, r1, r2	Один из регистров А, В, С, D, Е, H, L
DDD, SSS	Биты, адресующие один из регистров А, В, С, D,

Е, Н, L (DDD — принимающий регистр, SSS — передающий регистр):

DDD или SSS ИМЯ РЕГИСТРА

111	A
000	B
001	C
010	D
011	E
100	H
101	L

gr Одна из регистровых пар: символ В задает пару регистров В, С. При этом регистр В является старшим, а С — младшим
символ D задает пару регистров D, E. Регистр D является старшим, а E — младшим
Символ Н задает пару регистров Н, L. Регистр Н является старшим, а L — младшим
Символы SP задают 16-битный регистр-указатель стека

RP Биты, адресующие одну из регистровых пар В, D, Н, S, P:

RP	РЕГИСТРОВАЯ ПАРА
00	B, C
01	D, E
10	H, L
11	S, P

rh Первый (старший) регистр выбранной регистровой пары

rl Второй (младший регистр) выбранной регистровой пары

PC 16-битный счетчик команд (PCN и PCL используются для обозначения старшего и младшего байт адреса соответственно)

SP 16-битный регистр-указатель стека (SPH и SPL используются для обозначения старшего и младшего байт адреса соответственно)

rm Бит m регистра r (нумерация бит регистров — от 0 до 7 справа налево)

Z, S, P, CY, AC Признаки состояния: НУЛЬ, ЗНАК, ПАРИТЕТ, ПЕРЕНОС и ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС соответственно

() Содержимое ячейки памяти или регистров, символическое имя которых заключено в скобки

← Оператор пересылки

Λ Логическое И

∨ Исключающее ИЛИ

∇ Логическое ИЛИ

+ Сложение

— Вычитание в дополнительном коде

***** Умножение

↔ Оператор обмена

— Обратный код [например, (\bar{A})]

n Вектор прерывания (число от 0 до 7)

NNN Двоичный код вектора прерывания (от 000 до 111)

Формат описания

Последующий материал представляет собой детальное описание системы команд МП 8080. Каждая команда описывается следующим образом:

1. В начале первой строки полужирным шрифтом приводятся мнемокод операции и операнд команды по правилам языка ассемблера МАС 80.

2. Имя команды (раскрытие аббревиатуры) представляется в конце первой строки.

3. Следующая строка (строки) представляет собой символическое описание операции, выполняемой по данной команде.

4. В следующей строке (строках) содержится словесное описание операции, выполняемой по данной команде.

5. Следующая строка (строки) содержат двоичные коды и формат команды.

6. Последние четыре строки содержат информацию об особенностях выполнения команды. Прежде всего указывается число машинных циклов и состояний процессора. Если команда имеет два возможных времени выполнения, то, как например в случае условных переходов, указываются через дробь оба эти значения. В следующей строке описывается способ адресации и, наконец, в последней строке перечисляются признаки, изменяемые в процессе выполнения данной команды.

Группа команд пересылки

Группа команд пересылки реализует операции обмена данными между регистрами и памятью. Ни одна из команд данной группы не изменяет содержимое регистра признаков состояния.

MOV r1, r2 (Move Register)

(r1) ← (r2)

Содержимое регистра r2 передать в регистр r1.

0	1	D	D	D	S	S	S
---	---	---	---	---	---	---	---

Циклы : 1

Состояния : 5

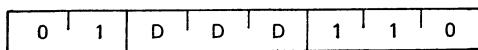
Адресация : регистровая

Признаки : отсутствуют

MOV r, M (Move from memory)

$(r) \leftarrow ((H)(L))$

Содержимое ячейки памяти, адрес которой содержится в регистрах H и L, передать в регистр r.



Циклы : 2

Состояния : 7

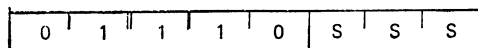
Адресация : косвенно-регистровая

Признаки : отсутствуют

MOV M, r (Move to memory)

$((H)(L)) \leftarrow (r)$

Содержимое регистра r передать в ячейку памяти, адресуемую содержимым регистров H и L.



Циклы : 2

Состояния : 7

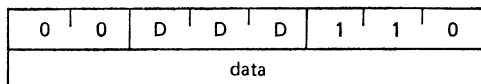
Адресация : косвенно-регистровая

Признаки : отсутствуют

MVI r, data (Move Immediate)

$(r) \leftarrow (\text{byte } 2)$

Содержимое второго байта команды передать в регистр r.



Циклы : 2

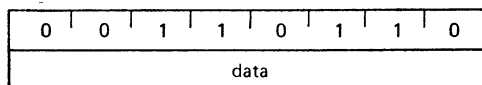
Состояния : 7

Адресация : непосредственная
 Признаки : отсутствуют

MVI M, data (Move to memory immediate)

$((H)(L)) \leftarrow (\text{byte } 2)$

Содержимое второго байта команды передать в ячейку памяти, адресуемой содержимым регистров H и L.



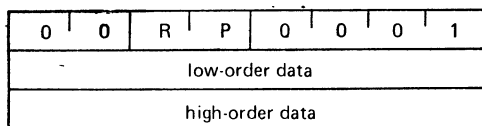
Циклы : 3
 Состояния : 10
 Адресация : непосредственная/косвенно-
 регистровая
 Признаки : отсутствуют

LXI rp, data 16 (Load register pair immediate)

$(rh) \leftarrow (\text{byte } 3)$

$(rl) \leftarrow (\text{byte } 2)$

Байт 3 команды передать в старший регистр регистровой пары гр. Байт 2 команды передать в младший регистр регистровой пары гр.



Циклы : 3
 Состояния : 10
 Адресация : непосредственная
 Признаки : отсутствуют

Группа команд арифметических операций

Данная группа команд выполняет арифметические операции с данными, находящимися в регистрах и памяти.

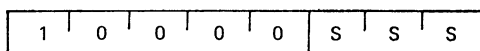
Все команды этой группы (за некоторым исключением) изменяют признаки состояния НУЛЬ, ЗНАК, ПАРИТЕТ, ПЕРЕНОС и ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС.

Все операции вычитания выполняются в дополнительных кодах, при этом признак ПЕРЕНОС выполняет функции признака ЗАЕМ.

ADD r (Add Register)

$$(A) \leftarrow (A) + (r)$$

Содержимое регистра сложить с содержимым аккумулятора. Результат операции разместить в аккумуляторе.

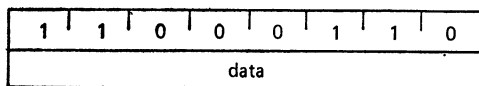


Циклы : 1
 Состояния : 4
 Адресация : регистровая
 Признаки : Z, S, P, CY, AC

ADI data (Add immediate)

$$(A) \leftarrow (A) + (\text{byte } 2)$$

Содержимое второго байта команды прибавить к содержимому аккумулятора. Результат операции разместить в аккумуляторе.



Циклы : 2
 Состояния : 7
 Адресация : непосредственная
 Признаки : Z, S, P, CY, AC

ADD M (Add memory)

$$(A) \leftarrow (A) + ((H) (L))$$

Содержимое ячейки памяти, адресуемой содержимым регистровой пары H, L, прибавить к содержимому аккумуля-

лятора. Результат операции сохранить в аккумуляторе.

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

Циклы : 2
 Состояния : 7
 Адресация : косвенно-регистравая
 Признаки : Z, S, P, CY, AC

ADC r (Add Register with carry)

$$(A) \leftarrow (A) + (r) + (CY)$$

Содержимое регистра и признака переноса сложить с содержимым аккумулятора. Результат операции разместить в аккумуляторе.

1	0	0	0	1	S	S	S
---	---	---	---	---	---	---	---

Циклы : 1
 Состояния : 4
 Адресация : регистровая
 Признаки : Z, S, P, CY, AC

ADC M (Add memory with carry)

$$(A) \leftarrow (A) + ((H) (L)) + (CY)$$

Содержимое ячейки памяти, адресуемой регистровой парой H, L, сложить с содержимым аккумулятора и признака переноса. Результат операции разместить в аккумуляторе.

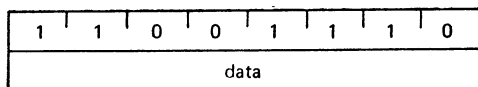
1	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---

Циклы : 2
 Состояния : 7
 Адресация : косвенно-регистравая
 Признаки : Z, S, P, CY, AC

ACI data (Add immediate with carry)

$$(A) \leftarrow (A) + (\text{byte } 2) + (CY)$$

Содержимое второго байта команды сложить с содержимым аккумулятора и признака переноса (CY). Результат операции разместить в аккумуляторе.

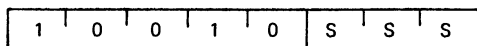


Циклы : 2
 Состояния : 7
 Адресация : непосредственная
 Признаки : Z, S, P, CY, AC

SUB r (Subtract Register)

$$(A) \leftarrow (A) - (r)$$

Содержимое регистра r вычесть из содержимого аккумулятора. Результат операции разместить в аккумуляторе.

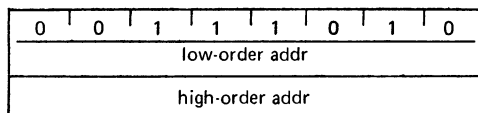


Циклы : 1
 Состояния : 4
 Адресация : регистровая
 Признаки : Z, S, P, CY, AC

LDA addr (Load Accumulator direct)

$$(A) \leftarrow ((\text{byte } 3) (\text{byte } 2))$$

Содержимое ячейки памяти, адресуемой во втором и третьем байтах команды, передать в регистр A.

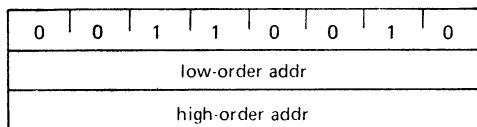


Циклы : 4
 Состояния : 13
 Адресация : прямая
 Признаки : отсутствуют

STA addr (Store Accumulator direct)

$((\text{byte } 3) (\text{byte } 2)) \leftarrow (A)$

Содержимое аккумулятора передать в ячейку памяти, адресованную во втором и третьем байтах команды.



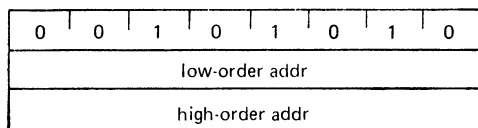
Циклы : 4
 Состояния : 13
 Адресация : прямая
 Признаки : отсутствуют

LHLD addr (Load H and L direct)

$(L) \leftarrow ((\text{byte } 3) (\text{byte } 2))$

$(H) \leftarrow ((\text{byte } 3) (\text{byte } 2) + 1)$

Содержимое ячейки памяти, адресуемой во втором и третьем байтах команды, передать в регистр L. Содержимое ячейки памяти по следующему адресу передать в регистр H.



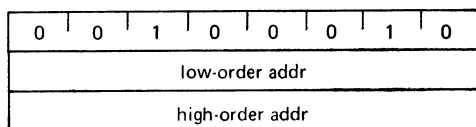
Циклы : 5
 Состояния : 16
 Адресация : прямая
 Признаки : отсутствуют

SHLD addr (Store H and L direct)

$((\text{byte } 3) (\text{byte } 2)) \leftarrow (L)$

$((\text{byte } 3) (\text{byte } 2) + 1) \leftarrow (H)$

Содержимое регистра L передать в ячейку памяти, адресуемую во втором и третьем байтах. Содержимое регистра H передать в ячейку памяти с последующим адресом.

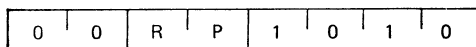


Циклы : 5
 Состояния : 16
 Адресация : прямая
 Признаки : отсутствуют

LDAX rp (Load accumulator indirect)

$(A) \leftarrow ((rp))$

Содержимое ячейки памяти, адресуемой в регистровой паре rp, передать в аккумулятор. **Примечание.** Могут использоваться только регистровые пары В, С и D, Е.



Циклы : 2
 Состояния : 7
 Адресация : косвенно-регистровая
 Признаки : отсутствуют

STAX rp (Store accumulator indirect)

$((rp)) \leftarrow (A)$

Содержимое аккумулятора передать в ячейку памяти, адресуемой регистровой парой rp. **Примечание.**

Могут использоваться только регистровые пары В, С или D, Е.

0	0	R	P	0	0	1	0
---	---	---	---	---	---	---	---

Циклы : 2
 Состояния : 7
 Адресация : косвенно-регистровая
 Признаки : отсутствуют

XCHG (Exchange H and L with D and E)

$(H) \leftrightarrow (D)$

$(L) \leftrightarrow (E)$

Содержимое регистров H и L взаимно обменять с содержимым регистров D и E.

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

Циклы : 1
 Состояния : 4
 Адресация : регистровая
 Признаки : отсутствуют

SUB M (Subtract memory)

$(A) \leftarrow (A) - ((H)(L))$

Содержимое ячейки памяти, адресуемой регистровой парой H, L, вычесть из содержимого аккумулятора. Результат поместить в аккумулятор.

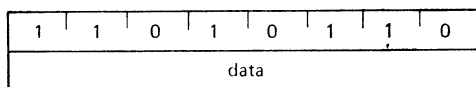
1	0	0	1	0	1	1	0
---	---	---	---	---	---	---	---

Циклы : 2
 Состояния : 7
 Адресация : косвенно-регистровая
 Признаки : Z, S, P, CY, AC

SUI data (Subtract immediate)

$$(A) \leftarrow (A) - (\text{byte } 2)$$

Содержимое второго байта команды вычесть из содержимого аккумулятора. Результат поместить в аккумулятор.

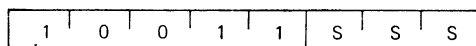


Циклы : 2
Состояния : 7
Адресация : непосредственная
Признаки : Z, S, P, CY, AC

SBB r (Subtract Register with borrow)

$$(A) \leftarrow (A) - (r) - (CY)$$

Содержимое регистра r и значение займа из разряда CY регистра признаков вычесть из содержимого аккумулятора. Результат поместить в аккумулятор.

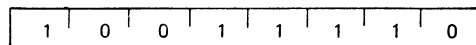


Циклы : 1
Состояния : 4
Адресация : регистровая
Признаки : Z, S, P, CY, AC

SBB M Subtract memory with borrow

$$(A) \leftarrow (A) - ((H) (L)) - (CY)$$

Содержимое ячейки памяти, адресуемой в регистровой паре H, L, и значение займа из разряда CY регистра признаков вычесть из содержимого аккумулятора. Результат поместить в аккумулятор.

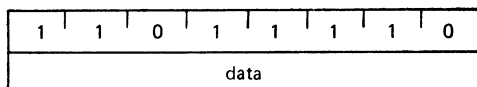


Циклы : 2
 Состояния : 7
 Адресация : косвенно-регистравая
 Признаки : Z, S, P, CY, AC

SBI data (Subtract immediate with borrow)

$$(A) \leftarrow (A) - (\text{byte } 2) - (CY)$$

Содержимое второго байта команды и значение займа из разряда CY регистра признаков вычесть из содержимого аккумулятора. Результат поместить в аккумулятор.

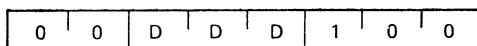


Циклы : 2
 Состояния : 7
 Адресация : непосредственная
 Признаки : Z, S, P, CY, AC

INR r (Increment register)

$$(r) \leftarrow (r) + 1$$

Содержимое регистра r увеличить на 1. **П р и м е ч а н и е.** Действуют все признаки, кроме признака переноса CY.



Циклы : 1
 Состояния : 5
 Адресация : регистровая
 Признаки : Z, S, P, AC

INR M (Increment memory)

$$((H) (L)) \leftarrow ((H) (L)) + 1$$

Содержимое ячейки памяти, адресуемой регистровой парой H, L, увеличить на 1. **П р и м е ч а н и е.** Действуют все признаки, кроме признака переноса CY.

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

Циклы : 3
 Состояния : 10
 Адресация : косвенно-регистровая
 Признаки : Z, S, P, AC

DCRr (Decrement Register)

$$(r) \leftarrow (r) - 1$$

Содержимое регистра r уменьшить на 1. **Примечание.** Действуют все признаки, кроме признака переноса CY.

0	0	D	D	D	1	0	1
---	---	---	---	---	---	---	---

Циклы : 1
 Состояния : 5
 Адресация : регистровая
 Признаки : Z, S, P, AC

DCR M (Decrement memory)

$$((H) (L)) \leftarrow ((H) (L)) - 1$$

Содержимое ячейки памяти, адресуемой регистровой парой H, L, уменьшить на 1. **Примечание.** Действуют все признаки, кроме признака переноса CY.

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

Циклы : 3
 Состояния : 10
 Адресация : косвенно-регистровая
 Признаки : Z, S, P, AC

INX rp (Increment register pair)

$$(rh) (rl) \leftarrow (rh) (rl) + 1$$

Содержимое регистровой пары *gr* увеличить на 1. П р и м е ч а н и е. Все признаки не действуют.

0	0	R	P	0	0	1	1
---	---	---	---	---	---	---	---

Циклы : 1
Состояния : 5
Адресация : регистровая
Признаки : отсутствуют

DCX *gr* (Decrement register pair)

$(rh)(rl) \leftarrow (rh)(rl) - 1$

Содержимое регистровой пары *gr* уменьшить на 1. П р и м е ч а н и е. Все признаки отсутствуют.

0	0	R	P	1	0	1	1
---	---	---	---	---	---	---	---

Циклы : 1
Состояния : 5
Адресация : регистровая
Признаки : отсутствуют

DAD *gr* (Add register pair to H and L)

$(H)(L) \leftarrow (H)(L) + (rh)(rl)$

Содержимое регистровой пары сложить с содержимым регистровой пары H, L. Результат поместить в регистровую пару H, L. П р и м е ч а н и е. Действует только признак переноса CY. Он устанавливается в 1, если имеет место перенос при суммировании с двойной точностью; в остальных случаях он равен 0.

0	0	R	P	1	0	0	1
---	---	---	---	---	---	---	---

Циклы : 3
Состояния : 10
Адресация : регистровая
Признаки : CY

DAA (Decimal Adjust Accumulator)

Восьмибитное число в аккумуляторе рассматривается как две 4-битные десятичные двоично-кодированные цифры. При этом выполняется коррекция результата операции по следующим правилам:

1. Если значение младших 4 бит аккумулятора больше девяти или если признак вспомогательного переноса АС равен 1, то к содержимому аккумулятора добавляется число 6.

2. Если значение старших 4 бит аккумулятора больше десяти или если признак переноса равен 1, то к содержимому старших 4 бит аккумулятора добавляется число 6.

Примечание. Все признаки действуют.

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

Циклы : 1

Состояния : 4

Признаки : Z, S, P, CY, AC

Группа команд логических операций

Данная группа команд выполняет булевы операции над содержимым регистров, ячеек памяти и регистра признаков.

Если это не оговаривается специально, то все команды данной группы оперируют с признаками НУЛЬ, ЗНАК, ПАРИТЕТ, ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС и ПЕРЕНОС обычным образом.

ANA r (AND Register)

$(A) \leftarrow (A) \wedge (r)$

Выполнить операцию логического И над содержимым регистра и аккумулятора. Результат поместить в аккумулятор. Признак переноса CY устанавливается в 0.

1	0	1	0	0	S	S	S
---	---	---	---	---	---	---	---

Циклы : 1

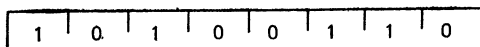
Состояния : 4

Адресация : регистровая
Признаки : Z, S, P, CY, AC

ANA M (AND memory)

$(A) \leftarrow (A) \wedge ((H) (L))$

Выполнить операцию логического И над содержимым аккумулятора и ячейки памяти, адресуемой содержимым регистровой пары H, L. Результат поместить в аккумулятор. Признак переноса CY устанавливается в 0.

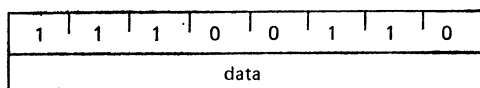


Циклы : 2
Состояния : 7
Адресация : косвенно-регистровая
Признаки : Z, S, P, CY, AC

ANI data (AND immediate)

$(A) \leftarrow (A) \wedge (\text{byte } 2)$

Произвести операцию логического И над содержимым второго байта команды и содержимым аккумулятора. Результат поместить в аккумулятор. Признаки CY и AC устанавливаются в 0.



Циклы : 2
Состояния : 7
Адресация : непосредственная
Признаки : Z, S, P, CY, AC

XRA r (Exclusive OR Register)

$(A) \leftarrow (A) \vee (r)$

Произвести операцию ИСКЛЮЧАЮЩЕЕ ИЛИ над содержимым регистра r и содержимым аккумулятора. Ре-

зультат поместить в аккумулятор. Признаки CY и AC устанавливаются в 0.

1	0	1	0	1	S	S	S
---	---	---	---	---	---	---	---

Циклы : 1
 Состояния : 4
 Адресация : регистровая
 Признаки : Z, S, P, CY, AC

XRA M (Exclusive OR Memory)

$(A) \leftarrow (A) \nabla ((H) (L))$

Произвести операцию ИСКЛЮЧАЮЩЕЕ ИЛИ над содержимым аккумулятора и содержимым ячейки памяти, адрес которой записан в регистровой паре H, L. Результат поместить в аккумулятор. Признаки CY и AC устанавливаются в 0.

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

Циклы : 2
 Состояния : 7
 Адресация : косвенно-регистровая
 Признаки : Z, S, P, CY, AC

XRI data (Exclusive OR immediate)

$(A) \leftarrow (A) \nabla (\text{bute } 2)$

Произвести операцию ИСКЛЮЧАЮЩЕЕ ИЛИ над содержимым второго байта команды и содержимым аккумулятора. Результат поместить в аккумулятор. Признаки CY и AC устанавливаются в 0.

1	1	1	0	1	1	1	0
data							

Циклы : 2
 Состояния : 7

Адресация : непосредственная
 Признаки : Z, S, P, CY, AC

ORA r (OR Register)

$$(A) \leftarrow (A) \vee (r)$$

Произвести операцию ИЛИ над содержимым регистра r и содержимым аккумулятора. Результат поместить в аккумулятор. Признаки CY и AC устанавливаются в 0.

1	0	1	1	0	S	S	S
---	---	---	---	---	---	---	---

Циклы : 1
 Состояния : 4
 Адресация : регистровая
 Признаки : Z, S, P, CY, AC

ORA M (OR memory)

$$(A) \leftarrow (A) \vee ((H) (L))$$

Произвести операцию ИЛИ над содержимым аккумулятора и содержимым ячейки памяти, адрес которой записан в регистровой паре H, L. Результат поместить в аккумулятор. Признаки CY и AC устанавливаются в 0.

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

Циклы : 2
 Состояния : 7
 Адресация : косвенно-регистровая
 Признаки : Z, S, P, CY, AC

ORI data (OR Immediate)

$$(A) \leftarrow (A) \vee (\text{byte } 2)$$

Произвести операцию ИЛИ над содержимым второго байта команды и содержимым аккумулятора. Результат

поместить в аккумулятор. Признаки CY и AC устанавливаются в 0.

1	1	1	1	0	1	1	0
data							

Циклы : 2
 Состояния : 7
 Адресация : непосредственная
 Признаки : Z, S, P, CY, AC

CMP r (Compare Register)

$(A) - (r)$

Содержимое регистра r вычесть из содержимого аккумулятора. Содержимое аккумулятора остается без изменений. Регистр признаков состояния устанавливается в зависимости от результата вычитания. Признак Z устанавливается в 1, если $(A) = (r)$. Признак CY устанавливается в 1, если $(A) < (r)$.

1	0	1	1	1	S	S	S
---	---	---	---	---	---	---	---

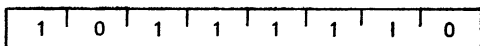
Циклы : 1
 Состояния : 4
 Адресация : регистровая
 Признаки : Z, S, P, CY, AC

CMP M (Compare memory)

$(A) - ((H)(L))$

Содержимое ячейки памяти, адрес которой находится в регистровой паре H, L, вычесть из содержимого аккумулятора. Содержимое аккумулятора остается без изменений. Регистр признаков состояний устанавливается в зависимости от результатов вычитания. Признак Z устанавли-

вается в 1, если $(A) = ((H) (L))$. Признак CY устанавливается в 1, если $(A) < ((H) (L))$.

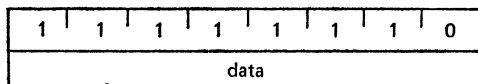


Циклы : 2
 Состояния : 7
 Адресация : косвенно-регистравая
 Признаки : Z, S, P, CY, AC

CPI data (Compare immediate)

$(A) - (\text{byte } 2)$

Содержимое второго байта команды вычесть из содержимого аккумулятора. Регистр признаков устанавливается в зависимости от результата вычитания. Признак Z устанавливается в 1, если $(A) = (\text{byte } 2)$. Признак CY устанавливается в 1, если $(A) < (\text{byte } 2)$.



Циклы : 2
 Состояния : 7
 Адресация : непосредственная
 Признаки : Z, S, P, CY, AC

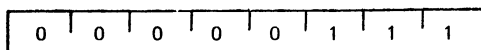
RLC (Rotate left)

$(A_{n+1}) \leftarrow (A_n); (A_0) \leftarrow (A_7)$

$(CY) \leftarrow (A_7)$

Содержимое аккумулятора сдвигается циклически на один разряд влево. Нулевой разряд содержимого аккумулятора и разряд CY регистра признаков приобретают значение старшего (седьмого) разряда аккумулятора, которое

он имел до сдвига. Действует только признак CY.



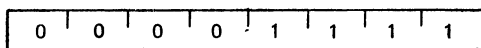
Циклы : 1
Состояния : 4
Признаки : CY

RRC (Rotate right)

$$(A_n) \leftarrow (A_{n-1}); (A_7) \leftarrow (A_0)$$

$$(CY) \leftarrow (A_0)$$

Содержимое аккумулятора циклически сдвинуть на один разряд вправо. Старший разряд аккумулятора и разряд CY регистра признаков приобретают значение младшего (нулевого) разряда аккумулятора, которое он имел до сдвига. Действует только признак CY.



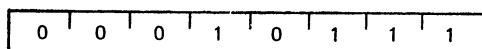
Циклы : 1
Состояния : 4
Признаки : CY

RAL (Rotate left through carry)

$$(A_{n+1}) \leftarrow (A_n); (CY) \leftarrow (A_7)$$

$$(A_0) \leftarrow (CY)$$

Содержимое аккумулятора циклически сдвинуть влево на один разряд, включая разряд CY регистра признаков. Младший разряд аккумулятора становится равным значению признака CY, а разряд CY регистра признаков приобретает значение старшего разряда аккумулятора. Действует только признак CY.



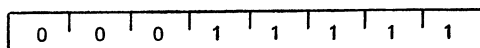
Циклы : 1
 Состояния : 4
 Признаки : CY

RAR (Rotate right through carry)

$$(A_n) \leftarrow (A_{n+1}); (CY) \leftarrow (A_0)$$

$$(A_7) \leftarrow (CY)$$

Содержимое аккумулятора циклически сдвинуть вправо на один разряд, включая разряд CY регистра признаков. Старший разряд аккумулятора становится равным значению признака CY, а разряд CY регистра признаков приобретает значение младшего разряда аккумулятора. Действует только признак CY.

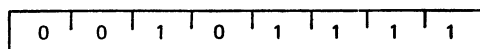


Циклы : 1
 Состояния : 4
 Признаки : CY

CMA (Complement accumulator)

$$(A) \leftarrow (\bar{A})$$

Инвертировать содержимое аккумулятора (нули становятся единицами, единицы — нулями). Признаки не действуют.



Циклы : 1
 Состояния : 4
 Признаки : отсутствуют

CMS (Complement carry)

$$(CY) \leftarrow (\overline{CY})$$

Инвертировать значение разряда CY регистра признаков. Остальные признаки не действуют.

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

Циклы : 1
Состояния : 4
Признаки : CY

STC (Set carry)

(CY) ← 1

Значение признака переноса CY установить в 1. Остальные признаки не действуют.

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

Циклы : 1
Состояния : 4
Признаки : CY

Группа команд передачи управления

Назначение команд этой группы — изменение нормального последовательного хода программы.

Команды этой группы не изменяют состояния разрядов регистра признаков.

Команды передачи управления бывают двух типов: условные и безусловные. Безусловные команды передачи управления выполняют определенные операции с содержимым регистра РС (счетчика команд). Команды условной передачи управления определяют состояние одного из четырех признаков регистра признаков и в зависимости от этого осуществляют переход к той или иной части программы. Условия эти следующие:

Условия	ССБ
NZ — не ноль ($Z=0$)	000
Z — ноль ($Z=1$)	001
NC — нет переноса ($CY=0$)	010
C — перенос ($CY=1$)	011
PO — нечетность ($P=0$)	100
PE — четность ($P=1$)	101
P — плюс ($S=0$)	110
M — минус ($S=1$)	111

JMP addr (Jump)

$(PC) \leftarrow (\text{byte 3}) (\text{byte 2})$

Передать управление команде, адрес которой определяется третьим и вторым байтами текущей команды.

1	1	0	0	0	0	1	1
low-order addr							
high-order addr							

Циклы : 3
 Состояния : 10
 Адресация : непосредственная
 Признаки : отсутствуют

J condition addr (Conditional jump)

If (CCC)

$(PC) \leftarrow (\text{byte 3}) (\text{byte 2})$

Если условие истинно, то адрес перехода по программе определяется содержимым второго и третьего байтов команды; в противном случае продолжается выполнение программы.

1	1	C	C	C	0	1	0
low-order addr							
high-order addr							

Циклы : 3
 Состояния : 10

Адресация : непосредственная
Признаки : отсутствуют

CALL addr (Call)

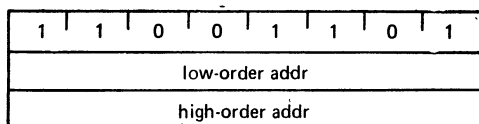
$((SP) - 1) \leftarrow (PCH)$

$((SP) - 2) \leftarrow (PCL)$

$(SP) \leftarrow (SP) - 2$

$(PC) \leftarrow (\text{byte } 3) (\text{byte } 2)$

Старшие восемь разрядов адреса очередной команды загружаются в ячейку памяти, адрес которой на единицу меньше содержимого регистра SP. Младшие восемь разрядов адреса очередной команды засылаются в ячейку памяти, адрес которой на две единицы меньше содержимого регистра SP. Управление передается команде, адрес которой определяется вторым и третьим байтами текущей команды.



Циклы : 5

Состояния : 17

Адресация : непосредственная/косвенно-
регистровая

Признаки : отсутствуют

C condition addr (Condition call)

If (CCC)

$((SP) - 1) \leftarrow (PCH)$

$((SP) - 2) \leftarrow (PCL)$

$(SP) \leftarrow (SP) - 2$

$(PC) \leftarrow (\text{byte } 3) (\text{byte } 2)$

Если условие истинно, то действия определяются выполнением команды CALL (см. выше); в противном случае

продолжается выполнение программы.

1	1	C	C	C	1	0	0
low-order addr							
high-order addr							

Циклы : 3/5
 Состояния : 11/17
 Адресация : непосредственная/косвенно-
 регистровая
 Признаки : отсутствуют

RET (Return)

$(PCL) \leftarrow ((SP))$

$(PCH) \leftarrow ((SP) + 1)$

$(SP) \leftarrow (SP) + 2$

Содержимое ячейки памяти, адрес которой определяется содержимым регистра SP, заносится на место младших восьми разрядов в счетчик команд PC. Содержимое ячейки памяти, адрес которой на единицу больше содержимого регистра SP, заносится на место старших восьми разрядов в счетчик команд PC. Содержимое регистра SP увеличивается на 2.

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

Циклы : 3
 Состояния : 10
 Адресация : косвенно-регистровая
 Признаки : отсутствуют

R condition (Condition return)

If (CCC)

$(PCL) \leftarrow ((SP))$

$(PCH) \leftarrow ((SP) + 1)$

$(SP) \leftarrow (SP) + 2$

Если условие истинно, то действия определяются выполнением команды RET (см. выше); в противном случае продолжается выполнение программы.

1	1	с	с	с	0	0	0
---	---	---	---	---	---	---	---

Циклы : 1/3
 Состояния : 5/11
 Адресация : косвенно-регистровая
 Признаки : отсутствуют

RST n (Restart)

$((SP) - 1) \leftarrow (PCH)$

$((SP) - 2) \leftarrow (PCL)$

$(SP) \leftarrow (SP) - 2$

$(PC) \leftarrow 8 * (NNN)$

Восемь старших разрядов адреса очередной команды заносятся в ячейку памяти, адрес которой на единицу меньше содержимого регистра SP. Восемь младших разрядов адреса очередной команды заносятся в ячейку памяти, адрес которой меньше содержимого регистра SP на 2. Содержимое регистра SP уменьшается на 2. Управление передается команде, адрес которой определяется как произведение содержимого поля NNN команды на 8.

1	1	N	N	N	1	1	1
---	---	---	---	---	---	---	---

Циклы : 3
 Состояния : 11
 Адресация : косвенно-регистровая
 Признаки : отсутствуют

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	N	N	N	0	0	0

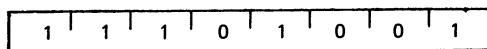
Содержимое счетчика команд после команды RST.

PCHL (Jump H and L indirect — move H and L to PC)

$(PCH) \leftarrow (H)$

$(PCL) \leftarrow (L)$

Содержимое регистра H занести в восемь старших разрядов счетчика команд PC. Содержимое регистра L занести в восемь младших разрядов команд PC.



Циклы : 1

Состояния : 5

Адресация : регистровая

Признаки : отсутствуют

Группа команд управления стеком, вводом-выводом и состояниями МП

Данная группа команд предназначена для выполнения операций ввода-вывода, операций с обращением к стеку, а также для управления внутренними признаками МП.

Признаки состояния не изменяются командами этой группы, если это не оговаривается специально.

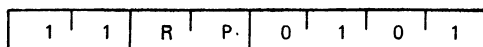
PUSH *гп* (Push)

$((SP) - 1) \leftarrow (rh)$

$((SP) - 2) \leftarrow (rl)$

$(SP) \leftarrow (SP) - 2$

Содержимое старшего регистра регистровой пары *гп* помещается в ячейку памяти, адрес которой на единицу меньше содержимого регистра SP. Содержимое младшего регистра регистровой пары *гп* помещается в ячейку памяти, адрес которой меньше содержимого регистра SP на 2. **П р и м е ч а н и е.** Регистровая пара *гп*-SP не может быть использована в данной команде.



Циклы : 3
 Состояния : 11
 Адресация : косвенно-регистравая
 Признаки : отсутствуют

PUSH PSW (Push processor status word)

$((SP) - 1) \leftarrow (A)$
 $((SP) - 2)_0 \leftarrow (CY); ((SP) - 2)_1 \leftarrow 1$
 $((SP) - 2)_2 \leftarrow (P); ((SP) - 2)_3 \leftarrow 0$
 $((SP) - 2)_4 \leftarrow (AC); ((SP) - 2)_5 \leftarrow 0$
 $((SP) - 2)_6 \leftarrow (Z); ((SP) - 2)_7 \leftarrow (S)$
 $(SP) \leftarrow (SP) - 2$

Содержимое регистра А засылается в ячейку памяти, адрес которой на единицу меньше содержимого регистра SP. Содержимое регистра признаков помещается в ячейку памяти, адрес которой меньше содержимого регистра SP на 2. Содержимое регистра SP уменьшается на 2.

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

Циклы : 3
 Состояния : 11
 Адресация : косвенно-регистравая
 Признаки : отсутствуют
СОДЕРЖИМОЕ РЕГИСТРА ПРИЗНАКОВ

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
S	Z	0	AC	0	P	1	CY

POP gp (Pop)
 $(r1) \leftarrow ((SP))$
 $(rh) \leftarrow ((SP) + 1)$
 $(SP) \leftarrow (SP) + 2$

Содержимое ячейки памяти, адрес которой определяется содержимым регистра SP, засылается в младший регистр регистровой пары гр. Содержимое ячейки памяти, адрес

которой на единицу меньше содержимого регистра SP, засылается в старший регистр регистровой пары гр. Содержимое регистра SP увеличивается на 2. **П р и м е ч а н и е.** Регистровая пара гр-SP не может быть использована в данной команде.

1	1	R	P	0	0	0	1
---	---	---	---	---	---	---	---

Циклы : 3
 Состояния : 10
 Адресация : косвенно-регистровая
 Признаки : отсутствуют

POP PSW (Pop processor status word)

$$(CY) \leftarrow ((SP))_0$$

$$(P) \leftarrow ((SP))_2$$

$$(AC) \leftarrow ((SP))_4$$

$$(Z) \leftarrow ((SP))_6$$

$$(S) \leftarrow ((SP))_7$$

$$(A) \leftarrow ((SP) + 1)$$

$$(SP) \leftarrow (SP) + 2$$

Содержимое ячейки памяти, адрес которой определяется содержимым регистра SP, используется для восстановления содержимого регистра признаков. Содержимое ячейки памяти, адрес которой на единицу больше содержимого регистра SP, засылается в аккумулятор. Содержимое регистра SP увеличивается на 2.

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

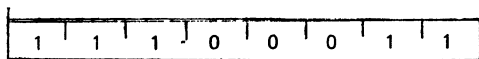
Циклы : 3
 Состояния : 10
 Адресация : косвенно-регистровая
 Признаки : Z, S, P, CY, AC

XTHL (Exchange stack top with H and L)

$(L) \leftrightarrow ((SP))$

$(H) \leftrightarrow ((SP) + 1)$

Содержимое регистра L обменивается содержимым с ячейкой памяти, адрес которой определяется содержимым регистра SP. Регистр H обменивается содержимым с ячейкой памяти, адрес которой на единицу больше содержимого регистра SP.



Циклы : 5

Состояния : 18

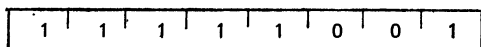
Адресация : косвенно-регистравая

Признаки : отсутствуют

SPHL (Move HL to SP)

$(SP) \leftarrow (H) (L)$

Содержимое регистров H и L (16 разрядов) занести в регистр SP.



Циклы : 1

Состояния : 5

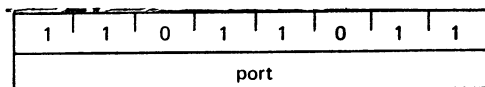
Адресация : регистравая

Признаки : отсутствуют

IN port (Input)

$(A) \leftarrow (data)$

Данные, выданные в двунаправленную шину данных адресуемому порту, передаются в аккумулятор.

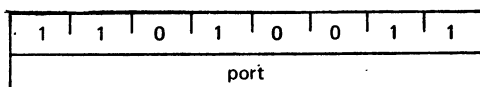


Циклы : 3
 Состояния : 10
 Адресация : прямая
 Признаки : отсутствуют

OUT port (Output)

(data) \leftarrow (A)

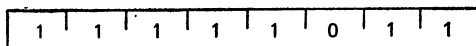
Содержимое аккумулятора выдается в двунаправленную шину данных для передачи в адресуемый порт.



Циклы : 3
 Состояния : 10
 Адресация : прямая
 Признаки : отсутствуют

EI (Enable interrupts)

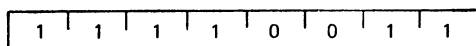
Разрешена работа системы прерывания после выполнения следующей команды.



Циклы : 1
 Состояния : 4
 Признаки : отсутствуют

DI (Disable interrupts)

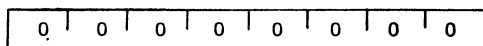
Запрещена работа системы прерывания непосредственно после данной команды.



Циклы : 1
 Состояния : 4
 Признаки : отсутствуют

(HLT) (Halt)

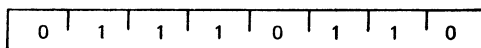
Процессор остановлен. Содержимое регистров и признаков не изменяется.



Циклы : 1
Состояния : 7
Признаки : отсутствуют

NOP (No op)

Пустая операция. Содержимое регистров и признаков не изменяется.



Циклы : 1
Состояния : 4
Признаки : отсутствуют

СПИСОК ЛИТЕРАТУРЫ, ДОБАВЛЕННОЙ ПРИ ПЕРЕВОДЕ

1. Вальков В. М. Микроэлектронные управляющие вычислительные комплексы. — Л.: Машиностроение, 1979. — 200 с.
2. Клингман Э. Проектирование микропроцессорных систем. — М.: Мир, 1980. — 575 с.
3. Каган Б. М., Сташин В. В. Микропроцессоры в цифровых системах. — М.: Энергия, 1979. — 193 с.
4. Прангишвили И. В. Микропроцессоры и микро-ЭВМ. — М.: Энергия, 1979. — 230 с.
5. Соучек Б. Микропроцессоры и микро-ЭВМ. — М.: Советское радио, 1979. — 517 с.
6. Вайда Ф., Чакань В. Микро-ЭВМ. — М.: Энергия, 1980 — 360 с.

СПИСОК ОСНОВНЫХ ТЕРМИНОВ И ОПРЕДЕЛЕНИЙ

Адрес — двухбайтное слово, используемое ЦП для указания ячейки памяти.

Адресация индексная представляет собой разновидность относительной адресации. При индексной адресации действительный адрес определяется путем сложения содержимого индексного регистра с базовым адресом, следующим за кодом операции.

Адресация косвенно-регистравая — способ адресации, при котором адрес ячейки памяти находится в специальном регистре ЦП, адресуемом в команде неявным образом.

Адресация непосредственная — способ адресации, при котором операнд находится в теле команды. Операнд хранится в ячейке памяти, которая следует непосредственно за ячейкой, в которой располагается код операции команды.

Адресация неявная (подразумеваемая) — способ адресации, при котором адрес одного из операндов неявно задается в коде операции. Чаще всего неявным способом адресуется содержимое регистра-аккумулятора.

Адресация относительная — способ адресации, при котором действительный адрес определяется путем суммирования адреса, содержащегося в команде, с числом, которое может быть или адресом данной команды, или адресом первой ячейки текущей страницы памяти, или содержанием одного из регистров ЦП.

Адресация прямая — способ адресации, при котором адрес операнда содержится в теле команды.

Аккумулятор — специальный регистр ЦП, который при выполнении арифметических и логических операций является источником одного из операндов и местом фиксации результата выполнения операции. Аккумулятор, кроме того, является неявно адресуемым регистром ЦП, который используется при выполнении операций ввода-вывода информации.

Ассемблер — кросс- или резидентная программа, предназначенная для трансляции исходной программы, записанной на языке ассемблера в объектную программу.

Байт — наименьшая адресуемая единица информации. В микро-ЭВМ чаще всего используются 8-разрядные байты.

Бит — двоичная цифра.

Бит паритета — бит регистра признаков, используемый для организации контроля передачи данных по четности или нечетности числа единиц в машинном слове.

Бод — единица скорости передачи двоичной информации последовательным кодом в секунду.

Вектор прерывания — трехразрядный код в теле команды ПО-ВТОРНЫЙ ЗАПУСК, который ЦП использует для формирования начального адреса одной из восьми возможных подпрограмм обслуживания запроса прерывания.

Вспомогательный перенос — признак результата выполнения операции в ЦП, который устанавливается в единицу в случае возникновения переноса из младшей тетрады байта в старшую. Признак используется при выполнении операций с десятичными двоично-кодированными числами.

Выбор модуля (корпуса, кристалла) — часть адреса памяти, указывающая, в каком модуле расположено данное слово (1); выбор, селектирующие входы корпуса БИС, предназначенные для выбора данного модуля из группы подобных (2).

Выборка — машинный цикл, в котором выполняется обращение к памяти и передача в ЦП байта, хранимого в адресуемой ячейке памяти.

Высокоимпедансное состояние — состояние «не 0, не 1» на выходе электронной схемы с тремя состояниями, при котором выход схемы отключается от нагрузки.

Дисплей линейный семисегментный — набор семисегментных светодиодных индикаторов, расположенных в ряд (на печатной плате или

на лицевой панели микро-ЭВМ) и предназначенных для отображения информации в восьмеричных или шестнадцатеричных кодах. Используется для визуального контроля вводимой с клавиатуры информации и для отображения содержимого регистров или ячеек памяти микро-ЭВМ.

Драйвер — электронная схема, выполняющая функции усилителя мощности.

Европлата — печатная плата для монтажа интегральных микросхем с геометрическими размерами 10×16 см.

Захват цикла — приостанов работы ЦП на один машинный цикл для передачи по системной шине байта данных в режиме прямого доступа к памяти.

Интерфейс ввода-вывода — программно-управляемые БИС сопряжения УВВ и ЦП в микро-ЭВМ.

Карта памяти — графическое изображение адресного поля памяти, на котором адреса машинных слов поставлены в соответствие функционально-различным зонам памяти (ОЗУ, ПЗУ, стек, область подпрограмм, область констант и т. п.).

Квити́рование — метод взаимодействия микро-ЭВМ с периферийным оборудованием, при котором между ними осуществляется обмен сигналами управления и сигналами состояния с целью взаимной синхронизации. Метод передачи данных с квитированием позволяет согласовать скорости выполнения операций в «медленных» УВВ и в «быстрых» ЦП.

Микромаши́нная система — вычислительная система, в состав которой входят микро-ЭВМ и набор периферийного оборудования для обеспечения связи микро-ЭВМ с пользователем.

Микропроцессор — одна или несколько БИС, предназначенных для обеспечения связи микро-ЭВМ с пользователем.

Микро-ЭВМ — вычислительная машина, в состав которой входят микропроцессор, память и интерфейс ввода-вывода информации.

Модуль памяти — одна или несколько БИС памяти, образующих конструктивную единицу хранения машинных слов.

Операционная система — комплекс служебных программ, обеспечивающих управление работой универсальной микро-ЭВМ, в том числе решение задач распределения адресов памяти и УВВ, обработки прерываний, планирования заданий и т. п.

Память — часть аппаратных средств микро-ЭВМ, в которой хранятся команды и данные. Каждому байту команд и данных ставится в соответствие адрес, который используется процессором при обращении к памяти.

Память оперативная — БИС памяти, при обращении к которой можно выполнить запись или считывание информации.

Память постоянная — БИС памяти, содержащей неизменяемую информацию.

Память постоянная программируемая — БИС памяти, в которую пользователь может однократно записать информацию при помощи специальной аппаратуры (программатора ППЗУ), а затем использовать ее как постоянную память.

Память постоянная репрограммируемая — БИС постоянной памяти, допускающая многократное занесение и стирание информации при помощи специальной аппаратуры.

«Плавающий потенциал» — высокоимпедансное состояние буферной схемы с тремя состояниями, при котором выходные цепи схемы отключаются от нагрузки.

Поддержка изготовителя — предоставление пользователю изготовителем микро-ЭВМ информации о способах применения микро-ЭВМ, наборов для создания прототипов, системного программного обеспечения и оказание технического содействия в разработке прикладных программ.

Порт ввода-вывода — адресуемый одно- или двунаправленный буферный регистр с выходом, отключаемым от нагрузки, предназначенный для построения простейшего программируемого интерфейса микро-ЭВМ с такими внешними устройствами, как клавиатура, линейный дисплей, фотосчитыватель и т. п.

Прерывание — механизм, обеспечивающий временное прекращение выполнения текущей программы и передачу управления специальной подпрограмме обслуживания прерывания.

Программа ассемблер — служебная программа, которая преобразует исходную программу, написанную на языке мнемокодов и символьческих адресов, в объектную. В процессе ассемблирования формируется список синтаксических ошибок, содержащихся в исходной программе, и выполняется распечатка исходной и объектной версий программы.

Программа-загрузчик — служебная программа для управления работой устройства ввода данных с перфоленты. Загрузчик обеспечивает размещение рабочей программы в требуемой области памяти и контролирует правильность расположения начального и конечного адресов программы в памяти микро-ЭВМ.

Программа-имитатор — служебная программа, позволяющая создавать рабочие программы для различных микропроцессоров с использованием единого программного комплекса разработки.

Программа исходная — программа работы микро-ЭВМ, написанная на символическом или алгебраическом языке.

Программа-компилятор — служебная программа, которая выполняет трансляцию программы, написанной на том или ином языке высокого уровня, в объектную программу.

Программа-монитор — служебная резидентная программа, предназначенная для управления работой микро-ЭВМ в процессе трансляции, тестирования, корректировки и ввода прикладных программ пользователя.

Программа объектная — программа на машинном языке, получаемая, как правило, в результате трансляции исходной программы.

Программа-отладчик — служебная программа, предназначенная для тестирования объектной программы. Отладчик обеспечивает разработчику следующие возможности: индикацию содержимого регистров и ячейку памяти, изменение содержимого памяти, останов по адресу, пошаговый режим выполнения программы и т. п.

Программа прикладная — упорядоченная последовательность команд для решения прикладной задачи.

Программа-редактор — служебная программа, используемая при подготовке исходной перфоленты. Редактор позволяет вносить исправления в текст программы пользователя после того, как она нанесена на перфоленту.

Программа-симулятор — моделирующая программа, используемая в процессе разработки прикладных программ пользователя и позволяющая моделировать работу проектируемой микропроцессорной системы на ЭВМ другого типа (кросс-ЭВМ).

Программное обеспечение (ПО) системное — набор сервисных программ, предназначенных для трансляции, редактирования, отладки и загрузки прикладных программ пользователя.

Программное обеспечение кроссовое (кросс-ПО) — служебные программы, предназначенные для разработки программ для микро-ЭВМ с помощью ЭВМ другого типа.

Программное обеспечение резидентное — набор служебных программ, реализованных в той же микро-ЭВМ, на которой будет работать прикладная программа пользователя.

Программно-аппаратное обеспечение — резидентное программное обеспечение, реализованное в БИС ПЗУ.

Программная совместимость микро-ЭВМ — качество, обеспечивающее переносимость пакетов прикладных программ (ППП) с одной микро-ЭВМ на другую. С точки зрения переносимости различают три группы ППП: библиотеку стандартных программ (БСП), автономные ППП и ППП, функционирующие в среде операционной системы.

Переносимость БСП осуществима в том случае, если микро-ЭВМ имеют единую систему команд и совместимость физических форматов и типов носителей информации.

Переносимость автономных ППП реализуется в том случае, если микро-ЭВМ имеют единую систему команд, совместимость физических форматов и типов носителей программ, а также единые архитектурные решения (карта памяти, адресация и назначение бит регистров устройств, организация системы прерываний, единый системный интерфейс микро-ЭВМ на логическом и физическом уровнях).

Переносимость ППП, функционирующих в операционной среде, обеспечивается в том случае, если микро-ЭВМ имеют единую систему команд, совместимость физических форматов и типов носителей программ, а также если в них используется единая операционная система и одинаковым способом обеспечивается выполнение системных стандартов и соглашений операционной системы.

Прототип — макет специализированной микропроцессорной системы, реализованной на основе прототип-набора, в состав которого входят печатные платы, кабели и разъемы, комплект микропроцессорных БИС, источник электропитания, переключатели и элементы индикации, БИС для наращивания объема памяти и часто БИС ПЗУ, в которой хранится монитор.

Распределение функций — этап проектирования микропроцессорной системы, на котором принимаются решения о разделении функций на аппаратно- и программно-реализуемые.

Регистр адреса — регистр ЦП, в котором фиксируется адрес ячейки памяти, к которой выполняется обращение.

Регистр буферный с тремя состояниями — регистр, выходные цепи которого могут быть установлены в высокоимпедансное состояние (отключены от нагрузки).

Регистр индексный — регистр общего назначения, используемый для модификации адреса.

Регистр признаков — совокупность триггеров хранения признаков результата выполнения операции в ЦП, таких как перенос, знак, ноль, паритет, вспомогательный перенос и т. п.

Регистр общего назначения — набор регистров ЦП, предназначенный для временного хранения данных, организации указателей памяти, индексной, относительной, косвенно-регистрационной адресации, счетчиков циклов и т. п. Регистры общего назначения образуют сверхоперативную память ЦП, для адресации которой используется укороченное адресное поле в команде.

Регистр — указатель данных — регистр ЦП, содержащий адреса данных и используемый командами с косвенно-регистрационной адресацией.

Регистр — указатель стекла фиксирует адрес, по которому выполнена последняя запись байта в стек, или адрес последующей записи в стек. Содержимое регистра—указателя стека декрементируется перед каждым обращением к стеку для записи байта и инкрементируется после каждого обращения к стеку для выборки байта.

Система команд — набор команд, которые выполняются данной микро-ЭВМ.

Система разработки — аппаратно-программный комплекс (специализированная микро-ЭВМ), предназначенный для ассемблирования (трансляции), редактирования и отладки прикладной программы пользователя. Система разработки может содержать аппаратные средства (набор конструкторов и комплектующих изделий) для создания прототипа микропроцессорной системы пользователя.

Слово состояния программы — содержимое аккумулятора и регистра признаков ЦП.

Стек — область памяти, адресуемая с использованием регистра — указателя стека. Обращение к стеку производится в соответствии с дисциплиной доступа «последний на входе — первый на выходе».

Страница памяти — упорядоченная группировка ячеек памяти по старшим разрядам адреса. В 8-разрядной микро-ЭВМ с 16-разрядной шиной адреса страницу образуют 256 последовательно расположенных байт, адреса которых отличаются только значением младшего байта.

Тренажер — одноплатная микро-ЭВМ, содержащая шестнадцатиричную клавиатуру, линейный семисегментный дисплей, монитор в ПЗУ емкостью от 1 до 4К и ОЗУ емкостью от 256 до 1К ячеек. Используется для изучения возможностей микро-ЭВМ и техники программирования. Может быть использована в качестве простейшей системы разработки прикладных программ.

Формат команды — число байт, необходимых для представления команды. Различают одно-, двух- и трехбайтные команды. Время выполнения команды зависит от ее формата.

Цикл — набор команд в программе, регулярно повторяющийся в неизменяемой последовательности.

Цикл команды — время, необходимое для выборки команды из памяти и ее исполнения.

Цикл машинный — временной отрезок работы ЦП в цикле команды, связанный с обращением ЦП к регистрам памяти или одному из устройств ввода-вывода информации.

Шина — группа линий передачи информации, объединенных общим функциональным признаком, например шина данных, шина адресов, шина управления.

Языковые средства — средства написания исходных программ, предоставляемые пользователю системным программным обеспечением (язык ассемблера в простейшем случае, один или несколько проблемно-ориентированных языков высокого уровня).

О Г Л А В Л Е Н И Е

Предисловие редактора перевода	3
Глава первая. Что такое ЭВМ	6
1.1. Введение	6
1.2. ЭВМ как электронное устройство	7
1.3. Как осуществляется обработка информации	8
1.4. Структурная схема ЭВМ	9
1.5. Память	11
1.6. Организация основной памяти	13
1.7. Устройства ввода-вывода	14
1.8. Блок-схема алгоритма	16
1.9. Языки программирования	19
1.10. Язык ассемблера	20
1.11. Программы-трансляторы	21
1.12. Шестнадцатиричная система счисления	22
Глава вторая. Что такое микро-ЭВМ	22
2.1. Введение	22
2.2. Организация микро-ЭВМ	23
2.3. Система шин микро-ЭВМ	28
2.4. Система соединения блоков микро-ЭВМ	28
2.5. Устройство управления	29
2.6. Счетчик команд	29
2.7. Выборка команды	29
2.8. Арифметическо-логическое устройство	31
2.9. Обмен данными в микро-ЭВМ	32
2.10. Структура микро-ЭВМ	34
Глава третья. Общие сведения о микро-ЭВМ	34
3.1. Введение	34
3.2. От идеи к объектной программе	36
3.3. От объектной программы к рабочей программе	38
3.4. Типы памяти	40
3.5. Пример применения микро-ЭВМ	42
3.6. Области применения микро-ЭВМ	44
3.7. Выбор типа микро-ЭВМ	44
Глава четвертая. Представление информации в ЭВМ	45
4.1. Введение	45
4.2. Десятичная система счисления	46
4.3. Двоичная система счисления	46
4.3.1. Преобразование чисел из двоичной системы в десятичную	48
4.4. Сложение двоичных чисел	48
4.5. Вычитание двоичных чисел	49
4.6. Дополнительный код двоичного числа	51
4.7. Умножение двоичных чисел	54

4.8. Терминология	54
4.9. Шестнадцатиричная система счисления	55
4.10. Двоично-десятичный код	56
4.11. Системы кодирования	56
Глава пятая. Схемотехника ЭВМ	59
5.1. Введение	59
5.2. Логический элемент И	60
5.3. Логический элемент ИЛИ	61
5.4. Инвертор	62
5.5. Логический элемент И-НЕ	63
5.6. Логический элемент ИЛИ-НЕ	64
5.7. Логический элемент ИСКЛЮЧАЮЩЕЕ ИЛИ	65
5.8. Триггеры	65
5.8.1. D-триггер	66
5.8.2. Двухтактный JK-триггер	67
5.9. Регистры	69
5.10. Компаратор	70
5.11. Сдвигающие регистры	71
5.11.1. Прием информации в регистр	71
5.11.2. Передача информации из регистра	72
5.11.3. Умножение и деление	72
5.12. Специальные схемы	73
5.12.1. Драйвер	73
5.12.2. Буферный регистр с тремя состояниями	73
Глава шестая. Основная память	77
6.1. Введение	77
6.2. Организация памяти	78
6.3. Длина слова	79
6.4. Модуль памяти	80
6.5. Адресация	82
6.6. Значение машинного слова	84
6.7. Пример	87
6.8. Цикл управления фон-Неймана	89
Глава седьмая. Введение в программирование	90
7.1. Введение	90
7.2. Шестнадцатиричные числа	91
7.3. Универсальная программа	92
7.4. Выполнение программы суммирования	93
7.5. Ввод программ и данных	97
7.6. Ввод данных с помощью переключателей	98
7.7. Ввод данных с шестнадцатиричной клавиатуры	99
7.8. Работа клавиатуры и дисплея	101
Глава восьмая. Архитектура центрального процессора. Часть 1	102
8.1. Введение	102
8.2. Арифметическо-логическое устройство	104
8.2.1. Арифметические операции	104
8.2.2. Логические операции	105
8.2.3. Взаимосвязь регистров и АЛУ	105
8.3. Регистр признаков	106
8.4. Признак переноса	107

8.5. Признак вспомогательного переноса	108
8.6. Признак нуля	109
8.7. Признак знака	109
8.8. Признак переполнения	110
8.9. Признак четности	110
8.10. Регистр команды и дешифратор кода операции	110
8.11. Регистры общего назначения	111
8.12. Счетчик команд	112
8.13. Указатель стека	113
8.13.1. Команды передачи управления	113
8.13.2. Подпрограммы	114
8.13.3. Указатель стека	116
8.14. Регистр адреса	116
Глава девятая. Архитектура центрального процессора.	117
Часть 2	
9.1. Генератор тактовых сигналов	117
9.2. Устройство управления	118
9.3. Временные диаграммы	119
9.4. Внешние выводы БИС микропроцессора	122
Глава десятая. Архитектура микро-ЭВМ	125
10.1. Введение	125
10.2. Структурная схема центрального процессора	125
10.3. Структурная схема памяти	126
10.4. Структурная схема модуля ввода-вывода	128
10.5. Синхронизация работы ЦП с памятью и УВВ	129
10.6. Потоки данных	130
10.6.1. Передать из регистра в регистр	131
10.6.2. Передать из памяти в регистр	131
10.6.3. Ввести данные	131
10.6.4. Выборка команды	132
10.6.5. Пересылка из регистра в регистр (MOV r ₁ , r ₂)	133
10.6.6. Пересылка из памяти в регистр (MOV r, M)	134
10.6.7. Ввод данных (IN port)	135
Глава одиннадцатая. Система команд	138
11.1. Введение	138
11.2. Команды пересылок	138
11.3. Команды арифметических операций	141
11.4. Команды логических операций	144
11.5. Команды передачи управления	150
11.6. Специальные команды	153
11.7. Пример программы	153
Глава двенадцатая. Синтаксис и подпрограммы	155
12.1. Введение	155
12.2. Структура команд ассемблера	156
12.3. Печать команд ассемблера	159
12.4. Директивы ассемблера	161
12.5. Подпрограммы	163
12.6. Команды вызова подпрограмм	163
12.7. Команды возврата из подпрограмм	164
12.8. Команды управления стеком	164
12.9. Пример	165

Глава тринадцатая. Техника адресации	168
13.1. Введение	168
13.2. Непосредственная адресация	168
13.3. Прямая адресация	169
13.4. Косвенная адресация	171
13.5. Неявная адресация	173
13.6. Относительная адресация	173
Глава четырнадцатая. Блок-схема алгоритма	176
14.1. Введение	176
14.2. Символы, используемые в блок-схеме	176
14.3. Типы блок-схем	178
14.4. Задача	179
14.5. Задача сортировки	180
14.6. Системная блок-схема	182
14.7. Основная блок-схема	183
14.8. Детальная блок-схема	185
Глава пятнадцатая. От постановки задачи к решению	189
15.1. Введение	189
15.2. Описание задачи	189
15.3. Анализ задачи	190
15.4. Составление блок-схемы	191
15.5. Написание исходной программы	192
15.6. От исходной программы к решению	195
Глава шестнадцатая. Примеры программ	196
Глава семнадцатая. Контроллер светофора для управления движением транспорта	205
17.1. Введение	205
17.2. Анализ характеристик системы	206
17.3. Разработка блок-схемы алгоритма	210
17.4. Разработка интерфейса	212
17.5. Интерфейс микро-ЭВМ и выходного оборудования	212
17.6. Интерфейс микро-ЭВМ и системы датчиков	213
17.7. Написание программы	214
17.8. Комплексование аппаратных средств и программного обеспечения	219
Глава восемнадцатая. Системное программное обеспечение	220
18.1. Введение	220
18.2. Программа-монитор	221
18.3. Редактор текста	222
18.4. Программа-ассемблер	224
18.5. Отладчик	227
18.6. Имитатор	230
18.7. Компилятор	231
18.8. Кросс- и резидентное программное обеспечение	232
Глава девятнадцатая. Системы разработки	234
19.1. Введение	234
19.2. Типовой цикл разработки	234
19.3. Цикл разработки систем на основе микропроцессоров	235
19.4. Методы разработки программ	237
	327

19.5.	Разработка программ с использованием системы разделения времени	238
19.6	Разработка программ на собственной ЭВМ	238
19.7.	Разработка программ с использованием аппаратных средств разработки	240
19.8.	Организация системы разработки	241
19.9.	Устройства ввода-вывода	243
19.10.	Микромашинный тренажер	247
Глава двадцатая. Периферийные устройства		250
20.1.	Введение	250
20.2.	Устройства ввода	250
20.3.	Устройства вывода	254
20.4.	Устройства управления	256
20.5.	Внешние запоминающие устройства	259
Глава двадцать первая. Интерфейс ввода-вывода микропроцессорной системы		264
21.1.	Введение	264
21.2.	Программируемый ввод-вывод	267
21.3.	Программируемый ввод с квитированием	267
21.4.	Программируемый вывод с квитированием	270
21.5.	Недостатки программируемого ввода-вывода	272
21.6.	Ввод-вывод информации по прерыванию	273
21.7.	Вывод данных по прерыванию	273
21.8.	Ввод данных по прерыванию	274
21.9.	Вектор прерывания	276
21.10.	Прямой доступ к памяти	279
Приложение. Система команд микропроцессора 8080		280
Список литературы, добавленной при переводе		318
Список основных терминов и определений		318

МИКРО-ЭВМ

Редактор издательства **А. Н. Гусяцкая**
 Переплет художника **В. П. Осипова**
 Технический редактор **О. Н. Адашкина**
 Корректор **З. Б. Драновская**

ИБ № 3064

Сдано в набор 16.09.81. Подписано в печать 16.03.82. Формат 84×108¹/₃₂. Бумага типографская № 2. Гарн. шрифта литературная. Печать высокая. Усл. печ. л. 17,22. Уч.-изд. л. 18,24. Тираж 40 000 экз. Заказ 66. Цена 1 р. 50 к.

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

Набрано и сматрицировано в ордена Октябрьской Революции, ордена Трудового Красного Знамени Ленинградском производственно-техническом объединении «Печатный Двор» имени А. М. Горького Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 197136, Ленинград, П-136, Чкаловский пр., 15

Отпечатано в Ленинградской типографии № 6 ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» имени Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 193144, Ленинград, ул. Моисеенко, 10.

1 р. 50 к.

ЭНЕРГОИЗДАТ