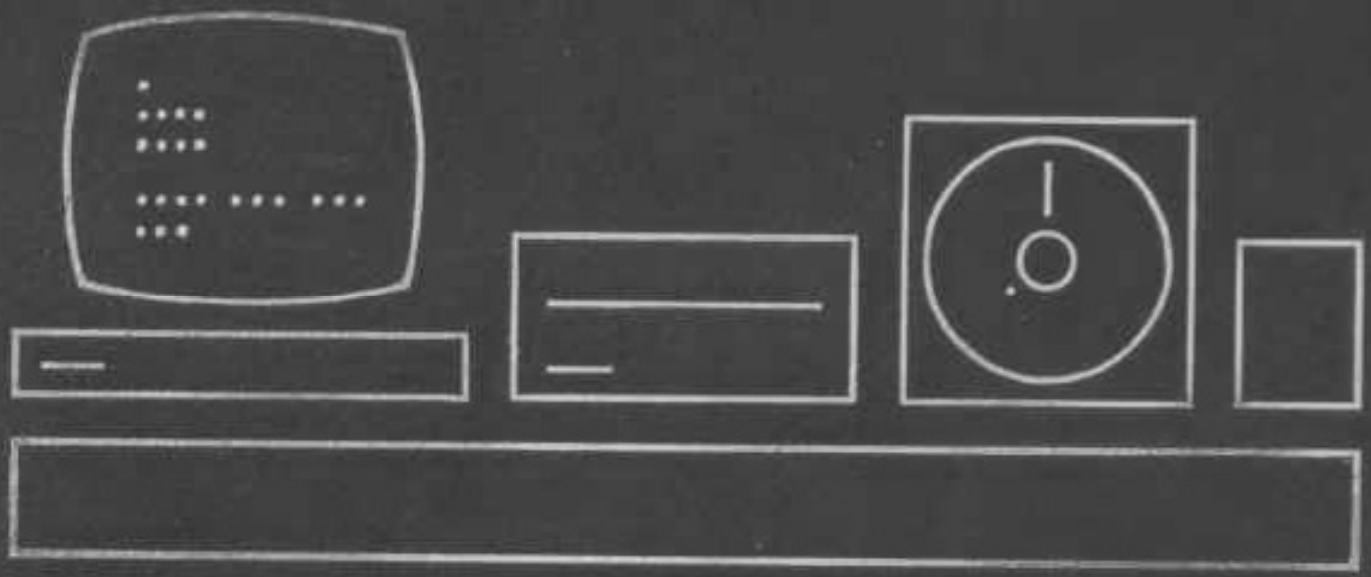
Ч. Гилмор

Введение в микропроцессорную технику



"Мир"

Ч. Гилмор

Введение в микропроцессорную технику

Перевод с английского
канд. техн. наук В. М. Кисельникова,
канд. техн. наук В. К. Потоцкого,
канд. техн. наук Л. В. Шабанова

Москва «Мир» 1984

ББК 32.973.2
Г 47
УДК 681.3

Гилмор Ч.

Г 47

Введение в микропроцессорную технику: Пер. с англ.–М.: Мир, 1984.–334 с., ил.

Книга американского специалиста является вводным курсом по микропроцессорам и микропроцессорным системам. Изложение ведется на примере гипотетической модели. Большое внимание уделено изучению программирования. Описана система команд микропроцессора и приведены примеры программ на языке ассемблера. Рассмотрены организация памяти микропроцессорных систем, основные методы адресации и традиционные средства ввода-вывода данных. Дано представление о технических средствах, применяемых при разработке и эксплуатации микропроцессорных систем.

Для специалистов, работающих в различных областях науки и техники, а также аспирантов и студентов старших курсов вузов.

$\Gamma \frac{2405000000\text{-}344}{041(01)\text{-}84}$ 158-84, ч. 1

ББК 32.973.2

Редакция литературы по новой технике

Предисловие к русскому изданию

С начала серийного производства микропроцессоров прошло более десяти лет. Это немалый срок для такой стремительно развивающейся области, как вычислительная техника, которая продолжает оказывать сильное влияние на все сферы человеческой деятельности. Однако литература на русском языке, в которой в доступной для массового читателя форме были бы систематизированы и обобщены современные представления о микропроцессорах, практически отсутствует. За последние несколько лет выпущен ряд изданий по принципам функционирования и проектирования, а также по применению микропроцессоров того или иного семейства[1]. Предпринимались также попытки сравнительного анализа возможностей различных семейств микропроцессоров[2]. Однако все эти книги либо относятся к тематически узконаправленным научным публикациям, либо к руководствам по эксплуатации микропроцессоров определенного типа. А это требует от читателя определенной подготовки в области вычислительной техники и ориентирует его на конкретный арсенал аппаратных и программных средств. Для тех же немногочисленных изданий, которые адресованы неподготовленному читателю, характерно сравнительно поверхностное изложение принципов построения микропроцессоров[3].

Такое положение дел нельзя признать удовлетворительным, если принять во внимание исключительно быстрое расширение сферы применения микропроцессорных средств вычислительной техники. Использование микропроцессоров позволяет достичь максимально возможного быстродействия самых разнообразных информационно-измерительных и управляющих систем при соблюдении требований экономии ресурсов. Поэтому все большее число инженерно-технических работников стремится не только получить фундаментальные представления по основам функционирования микропроцессорных систем, но и овладеть методикой их проектирования и навыками по эксплуатации.

Предлагаемая читателю книга является первой публикацией, освещающей весь комплекс проблем проектирования, функционирования и эксплуатации микропроцессорных систем «без привязки» к конкретному семейству микропроцессоров. Эта книга ориентирована на широкий круг лиц, имеющих лишь самое общее представление об основах электроники и схемотехники. При изложении материала автор использует гипотетический микропроцессор, обладающий типичными характеристиками микропроцессоров, выпускаемых в настоящее время. У этого микропроцессора отсутствуют особенности и «аномалии», вызывающие так много затруднений при первоначальном знакомстве с микропроцессорной техникой.

Книга отличается исключительно простым изложением материала и логической связностью отдельных частей. Автор на протяжении всей книги проводит принцип взаимосвязи аппаратных средств и программного обеспечения. Материал не только легко воспринимается, но и характеризуется высоким профессиональным уровнем изложения.

Выполнив все упражнения, рекомендованные автором, читатель сможет работать практически с любым современным микропроцессором.

Данная книга – хорошее учебное пособие по одному из наиболее важных разделов вычислительной техники. Нет сомнения, что она станет настольной книгой для многих инженерно-технических работников и студентов вузов.

В. К. Потоцкий

---

[1] См., например, Хилбурн Дж., Джулич П. Микро-ЭВМ и микропроцессоры: технические средства, программное обеспечение, применения: Пер. с англ.– М.: Мир, 1979; Соучек Б. Микропроцессоры и микро-ЭВМ: Пер. с англ.– М.: Советское радио, 1979.

[2] См., например, Соботка З., Стары Я. Микропроцессорные системы: Пер. с чеш.– М.: Энергоиздат, 1981.

[3] См., например, Морисуэ М., Ёсикава Т. Микро-ЭВМ за три дня: Пер. с япон.– М.: Мир, 1981.

ББК 32.973.2
Г 47
УДК 681.3

Гилмор Ч.

Г 47

Введение в микропроцессорную технику: Пер. с англ.– М.: Мир, 1984.– 334 с., ил.

Книга американского специалиста является вводным курсом по микропроцессорам и микропроцессорным системам. Изложение ведется на примере гипотетической модели. Большое внимание уделено изучению программирования. Описана система команд микропроцессора и приведены примеры программ на языке ассемблера. Рассмотрены организация памяти микропроцессорных систем, основные методы адресации и традиционные средства ввода-вывода данных. Дано представление о технических средствах, применяемых при разработке и эксплуатации микропроцессорных систем.

Для специалистов, работающих в различных областях науки и техники, а также аспирантов и студентов старших курсов вузов.

Г $\frac{2405000000\text{-}344}{041(01)\text{-}84}$ 158-84, ч. 1

ББК 32.973.2

Редакция литературы по новой технике

Предисловие к русскому изданию

С начала серийного производства микропроцессоров прошло более десяти лет. Это немалый срок для такой стремительно развивающейся области, как вычислительная техника, которая продолжает оказывать сильное влияние на все сферы человеческой деятельности. Однако литература на русском языке, в которой в доступной для массового читателя форме были бы систематизированы и обобщены современные представления о микропроцессорах, практически отсутствует. За последние несколько лет выпущен ряд изданий по принципам функционирования и проектирования, а также по применению микропроцессоров того или иного семейства[1]. Предпринимались также попытки сравнительного анализа возможностей различных семейств микропроцессоров[2]. Однако все эти книги либо относятся к тематически узконаправленным научным публикациям, либо к руководствам по эксплуатации микропроцессоров определенного типа. А это требует от читателя определенной подготовки в области вычислительной техники и ориентирует его на конкретный арсенал аппаратных и программных средств. Для тех же немногочисленных изданий, которые адресованы неподготовленному читателю, характерно сравнительно поверхностное изложение принципов построения микропроцессоров[3].

Такое положение дел нельзя признать удовлетворительным, если принять во внимание исключительно быстрое расширение сферы применения микропроцессорных средств вычислительной техники. Использование микропроцессоров позволяет достичь максимально возможного быстродействия самых разнообразных информационно-измерительных и управляющих систем при соблюдении требований экономии ресурсов. Поэтому все большее число инженерно-технических работников стремится не только получить фундаментальные представления по основам функционирования микропроцессорных систем, но и овладеть методикой их проектирования и навыками по эксплуатации.

Предлагаемая читателю книга является первой публикацией, освещающей весь комплекс проблем проектирования, функционирования и эксплуатации микропроцессорных систем «без привязки» к конкретному семейству микропроцессоров. Эта книга ориентирована на широкий круг лиц, имеющих лишь самое общее представление об основах электроники и схемотехники. При изложении материала автор использует гипотетический микропроцессор, обладающий типичными характеристиками микропроцессоров, выпускаемых в настоящее время. У этого микропроцессора отсутствуют особенности и «аномалии», вызывающие так много затруднений при первоначальном знакомстве с микропроцессорной техникой.

Книга отличается исключительно простым изложением материала и логической связностью отдельных частей. Автор на протяжении всей книги проводит принцип взаимосвязи аппаратных средств и программного обеспечения. Материал не только легко воспринимается, но и характеризуется высоким профессиональным уровнем изложения.

Выполнив все упражнения, рекомендованные автором, читатель сможет работать практически с любым современным микропроцессором.

Данная книга – хорошее учебное пособие по одному из наиболее важных разделов вычислительной техники. Нет сомнения, что она станет настольной книгой для многих инженерно-технических работников и студентов вузов.

В. К. Потоцкий

---

[1] См., например, Хилбурн Дж., Джулич П. Микро-ЭВМ и микропроцессоры: технические средства, программное обеспечение, применения: Пер. с англ. – М.: Мир, 1979; Соучек Б. Микропроцессоры и микро-ЭВМ: Пер. с англ. – М.: Советское радио, 1979.

[2] См., например, Соботка З., Стары Я. Микропроцессорные системы: Пер. с чеш. – М.: Энергоиздат, 1981.

[3] См., например, Морисуэ М., Ёсикава Т. Микро-ЭВМ за три дня: Пер. с япон. – М.: Мир, 1981.

Предисловие

Данная книга является вводным курсом по микропроцессорам и микро-ЭВМ. Она предназначена для тех, кто имеет самое общее представление об электронике. От читателя требуется сравнительно небольшая предварительная подготовка: знакомство с основами интегральных и цифровых электронных схем, а также владение базовым курсом высшей математики. В большинстве случаев материал излагается так, что достаточно лишь знаний в пределах двухсеместрового курса алгебры. Если же читатель имеет при этом некоторую подготовку в области схемотехники, то это существенно сокращает объем сведений, которые в противном случае ему пришлось бы принять «на веру».

По завершении изучения предлагаемого курса перед читателем открываются следующие возможности: специализироваться в области отладки и эксплуатации микропроцессорных систем либо продолжить освоение данного предмета, чтобы стать специалистом по проектированию и технологии производства упомянутых систем.

При изложении материала автор преследует две цели: дать возможность читателю получить представление о микропроцессорах как путем анализа микропрограмм и других программно-аппаратных средств, так и путем изучения аппаратных средств, т. е. электрических схем и механических конструкций. Значительное внимание уделяется базовому набору команд. В то же время ряд глав посвящен описанию столь важных компонентов аппаратной части, как память произвольного доступа, постоянная память, универсальный асинхронный приемопередатчик, устройства ввода-вывода и др.

Там, где это возможно, основные принципы поясняются примерами «реальной жизни». Обычно такие примеры имеют форму коротких программ решения задач сортировки, генерирования тактовых импульсов и т. д. Следует, однако, признать, что использование этих примеров не всегда уместно, поскольку, будучи зачастую лишь незначительной частью некоторой программы процессора, они являются логически не завершенными.

Излагаемый в книге материал состоит из трех частей. Гл. 1-5 посвящены основам микропроцессоров, в том числе принципам их построения, двоичной арифметике и основам программирования. В гл. 6-10 описываются базовый набор команд программного обеспечения и архитектура, типичные для большинства микропроцессоров. В гл. 11-14 обсуждаются аппаратные средства микропроцессоров и периферийных устройств, а также методы и средства проверки работоспособности микропроцессоров.

Рассмотрение материала проводится на базе некоего гипотетического микропроцессора. Это объясняется несколькими причинами. Прежде всего следует отметить, что большинство микропроцессоров, выпускаемых промышленностью, слишком сложно, чтобы служить основой для ознакомления с принципами построения и функционирования этих устройств. Однако овладевание указанными общими принципами делает доступным освоение микропроцессоров любого типа и модели. К тому же нет никаких оснований отдавать предпочтение в вводном курсе какому-либо конкретному микропроцессору или семейству. Более того, всегда есть опасность, что такой микропроцессор останется для изучающего наиболее предпочтительным. Что же касается выбранного здесь гипотетического микропроцессора, то некоторые его характеристики нельзя найти ни у одного современного микропроцессора. Прогнозировать же их у будущих моделей автор не берется, поскольку слишком быстро происходит в наше время смена техники и технологии.

Как обычно, в подготовке данного издания принимало участие много людей. Идеи, которые нашли отображение в книге, формировались в процессе бесчисленных дискуссий с талантливыми инженерами и преподавателями, связанными по роду деятельности с микропроцессорной техникой. С недостатками существующей методики обучения мы столкнулись при освоении микропроцессоров нашими инженерно-техническими работниками. В поисках рационального решения этой проблемы и возникла мысль о создании данной книги.

Хотелось бы выразить особую признательность Барбаре Титл и Дженис Браун за выполнение машинописных работ и корректуру текста. Я также благодарен моей жене Полли за помощь и поддержку в процессе редактирования текста и подготовки его к печати. Хотелось бы отметить вклад, внесенный Джози в составление примеров.

Заранее выражаю признательность читателям за все возможные замечания и предложения с их стороны.

Чарлз М. Гилмор

Рекомендации по технике безопасности

Электрические схемы и приборы могут представлять определенную опасность для пользователя. Необходимо принять соответствующие меры для предотвращения возможности возникновения пожара, взрыва, механических повреждений, электрического шока и ранений вследствие неправильной эксплуатации таких схем и приборов.

Наибольшую опасность, пожалуй, представляет собой электрический шок. При протекании через тело человека тока, превышающего 10 мА, оно может быть парализовано и лишено способности двигаться. А ведь 10 мА – это очень небольшой ток, всего лишь десять тысячных ампера. В обычном электрическом карманном фонаре протекает ток в 100 раз больший! Если человек подвергается действию тока более 100 мА, возникающее шоковое состояние может оказаться роковым, хотя такой ток все еще намного слабее тока карманного фонаря.

Батарея карманного фонаря может давать ток, достаточный, чтобы убить человека. Несмотря на это, фонарь безопасен в обращении, потому что сопротивление кожи человека обычно достаточно велико, чтобы ограничить силу протекающего по ней тока. Величина этого сопротивления, как правило, равна нескольким сотням тысяч ом. В низковольтных системах большое сопротивление ограничивает силу тока и удерживает его на уровне очень малых величин. Вот почему при работе с такими маломощными устройствами вероятность наступления электрического шока незначительна.

При наличии же высокого напряжения возможно возникновение тока, протекание которого сквозь кожу человека способно вызвать электрический шок. С увеличением напряжения растет вероятность шока, опасного для человека. Вот почему при работе с высоковольтными схемами необходимо использование специального оборудования и соблюдение определенных правил техники безопасности.

Если кожа человека влажная или на ней имеются порезы, ее сопротивление падает до нескольких сотен ом. В этом случае шок наступает при меньших значениях электрического напряжения. При наличии на коже ран даже напряжение 40 В может привести к роковому шоку. Однако, по мнению большинства инженерно-технических работников, 40 В – такое низкое напряжение, что нет необходимости в каких-либо мерах безопасности. Недооценка возможности серьезных последствий может оказаться губительной, поэтому следует быть очень осторожным даже при работе с так называемым низким напряжением.

Техника безопасности – это не только система знаний, но и определенная культура поведения. Персонал службы техники безопасности не может быть введен в заблуждение терминологией типа «низкое напряжение» или успокоен фактом функционирования средств защиты от нарушений правил техники безопасности. Нельзя считать обесточенной электрическую схему, даже если ключ питания находится в положении «Выключено». Ведь ключ может оказаться дефектным!

По мере накопления сведений об электричестве и электронике вы будете знакомиться с разнообразными правилами и средствами техники безопасности. А пока следуйте указанным ниже рекомендациям:

1) разберитесь, как устроена и функционирует схема (устройство), с которой вам предстоит работать, и только после этого приступайте к намеченным действиям;

2) следуйте предписываемой процедуре действий;

3) не приступайте к работе при наличии сомнений в правильности ваших действий; обратитесь за консультацией к специалисту.

Общие правила безопасности по электротехнике и электронике

Соблюдение мер безопасности гарантирует вам и тем, кто вас окружает, надежную защиту. Изучите правила, излагаемые ниже. Обсудите их с другими. Проконсультируйтесь у инструктора по технике безопасности по тем вопросам, которые вам не понятны.

1. Не работайте, если чувствуете усталость или сонливость (например вследствие принятия медикаментов).

2. Не работайте при плохом освещении.

3. Не работайте при повышенной влажности.

4. Используйте инструменты, приборы и защитные средства, предусмотренные правилами техники безопасности.

5. Не работайте во влажной одежде.

6. Снимайте перед работой кольца, браслеты и подобные металлические предметы вашего туалета.

7. Никогда не полагайте, что схема отключена. Проверяйте с помощью специального прибора или оборудования, все ли функционирует в соответствии с условиями работы.

8. Не пытайтесь внести изменения в работу средств безопасности. Не отсоединяйте систему блокировки. Удостоверьтесь, что она функционирует нормально.

9. Содержите ваш инструмент и оборудование в хорошем состоянии. Используйте инструменты, специально предназначенные для данного вида работ.

10 Удостоверьтесь, что конденсаторы разряжены. (Некоторые конденсаторы сохраняют заряд в течение продолжительного времени.)

11. Не отсоединяйте заземление. Убедитесь в том, что оно имеется во всех необходимых местах.

12. Не используйте приспособления, прерывающие цепи заземления.

13. Удостоверьтесь, что огнетушители, имеющиеся в вашем распоряжении, исправны. Вода способна проводить электрический ток и увеличивать размеры повреждений при возникновении пожара. Для борьбы с пожарами, возникающими при повреждении электрических цепей, наиболее предпочтительны галогеновые огнетушители и огнетушители, использующие двуокись углерода (CO_2). В некоторых случаях допустимо использование пенообразующих огнетушителей.

14. Соблюдайте правила пользования растворителями и другими химикалиями. Неправильное обращение с ними может привести к взрыву, воспламенению или повреждению электрических схем.

15. Некоторые электронные компоненты оказывают решающее влияние на безопасность функционирования оборудования. При замене одних компонентов другими следите за точным соответствием между ними.

16. Пользуйтесь защитной одеждой и специальными очками при работе с электровакуумными приборами, такими, как кинескоп телевизора.

17. Не пытайтесь работать со сложными схемами или оборудованием до получения соответствующей профессиональной подготовки.

18. Наиболее полная информация по технике безопасной эксплуатации электрического и электронного оборудования, как правило, находится в соответствующих руководствах, выпускаемых производителями этого оборудования. Пользуйтесь именно такой литературой!

Любая из приведенных здесь рекомендаций может быть расширена и дополнена. По мере изучения материала вы ознакомитесь более подробно с упомянутыми здесь рекомендациями. Помните, что информация по технике безопасности чрезвычайно важна и ею нужно овладеть на практике. От успешного решения этой проблемы зависит ваша жизнь.

Глава 1. Что такое микропроцессор?

Эта глава начинается кратким изложением истории создания микропроцессоров как результата достижений вычислительной техники и полупроводниковой технологии. Затем объясняется, что такое микропроцессор и каковы его две основные функции: обработка и управление. Вводится соответствующая терминология, которая хотя и лишена четких, научно обоснованных определений, закрепленных соответствующими стандартами, но поясняется применительно к основным понятиям.

Сравнение микропроцессора с электронными вычислительными машинами прошлого неизбежно. Поэтому объясняется разница между микропроцессором и микро-ЭВМ, хотя называть первый вычислительной машиной на одном кристалле – явное преувеличение. В заключение дается определение мощности микропроцессора посредством таких понятий, как размер слова данных, диапазон адресуемой памяти и скорость выполнения операций.

1.1. Краткий исторический обзор

Чтобы понять предпосылки появления микропроцессорных систем, нужно проследить развитие технологии производства цифровых вычислительных машин и полупроводниковых приборов. Ведь именно удачное сочетание достижений в указанных областях и привело в начале 70-х годов к созданию микропроцессоров.

Цифровая вычислительная машина выполняет расчеты под управлением *программы*. Общие принципы работы машины определяются ее *архитектурой*. Архитектура микропроцессора подобна архитектуре цифровой вычислительной машины. Иными словами, микропроцессор подобен цифровой машине, поскольку в обоих случаях вычисления выполняются под управлением программы. Следовательно, знакомство с историей создания цифровых вычислительных машин поможет нам лучше понять устройство и работу микропроцессоров. Столь же полезно знакомство с историческими сведениями, касающимися развития и совершенствования полупроводниковых схем, ибо микропроцессор – полупроводниковая схема, или точнее – большая интегральная микросхема.

Рис. 1.1 отображает основные этапы развития технологии производства цифровых машин и полупроводниковых схем на протяжении последних четырех десятилетий, т. е. со времен второй мировой войны, когда ЭВМ разрабатывали специально для военных целей. После войны, во второй половине 40-х годов, при проектировании машин преследовались главным образом научные и коммерческие интересы.

В период второй мировой войны значительные успехи были достигнуты и в технологии производства электронных схем. Радиолокация способствовала созданию быстродействующих цифровых схем, получивших название *импульсных*. Послевоенный период отмечен значительным

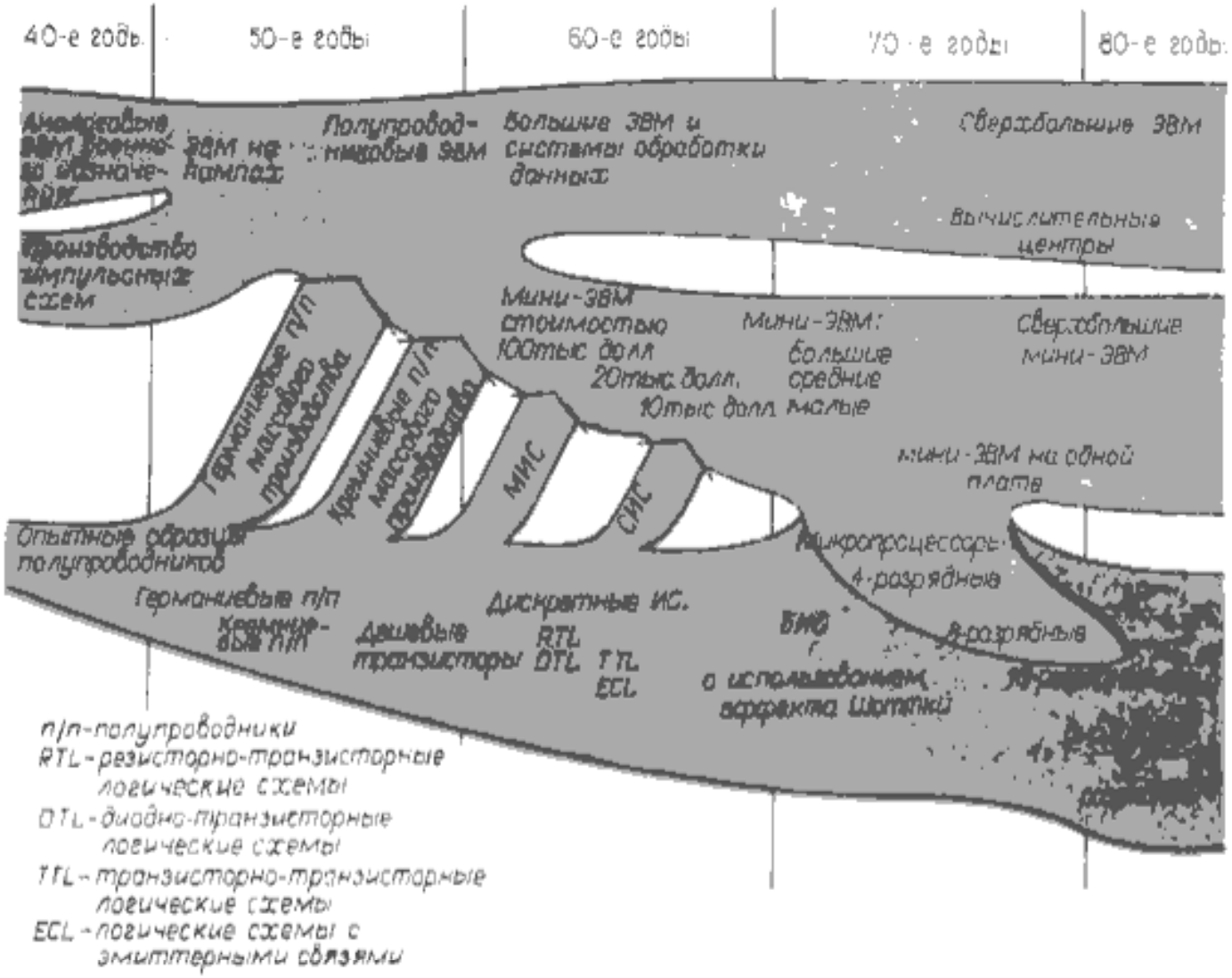


Рис. 1.1. Эволюция микропроцессора как продукта соединения двух технологий – производства вычислительных машин и полупроводниковых схем.

прогрессом в физике твердого тела: в 1948 г. ученые Научно-исследовательского центра Bell Laboratories изобрели транзистор.

Первые ЭВМ общего назначения появились в начале 50-х годов. Для построения основных логических элементов (вентилей и триггеров) в этих ЭВМ в качестве активных элементов использовались электровакуумные лампы. Из вентилей и триггеров собирались основные узлы машины, предназначенные для выполнения вычислений, управления и хранения информации. Электровакуумные лампы использовались также и для построения средств связи машины с «внешним» миром.

Лицам, знакомым с цифровыми схемами, известно, что для построения даже простого сумматора требуется значительное количество вентилей. Большинство цифровых машин содержит немалое число сумматоров и много других схем. Поскольку все эти схемы изготовлялись с использованием больших по объему ламп, первые цифровые машины были весьма громоздки. Лампы излучали тепло, и машины нуждались в системе охлаждения. Согласно современным стандартам, лампы тех лет были ненадежны. Как следствие этого, первые ЭВМ были дороги в производстве и эксплуатации. Недостатки электровакуумных приборов препятствовали совершенствованию цифровых вычислительных машин. Тем не менее уже в машинах того времени был заложен принцип *запоминаемой программы*.

В конце 40-х и начале 50-х годов вычислительные машины располагали для программирования *коммутационными панелями*. Манипулируя различными соединениями проводов, программист последовательно сообщал машине, что делать с данны-

ми. Кроме данных, в памяти не содержалось никакой информации. Позже, с реализацией принципа запоминаемой программы, последовательность операций над данными стала записываться в памяти в виде *слов* в цифровой форме. Единственным отличительным признаком этих слов от данных было их местоположение в памяти. Принцип запоминаемой программы явился важной фундаментальной концепцией, оказавшей влияние на архитектуру вычислительных машин.

50-е годы отмечены также бурным развитием технологии производства полупроводниковых схем. Углублялись представления о физике полупроводников. Использование кремния снизило затраты на производство схем, поскольку кремния в природе во много раз больше, чем германия – основного исходного материала в прошлом для изготовления полупроводников. Введение методов массового производства сделало транзисторы широко доступными и дешевыми.

Естественным было стремление разработчиков цифровых вычислительных машин заменить электровакуумные лампы транзисторами, и этот процесс начался в конце 50-х годов. Однако логические схемы, выполняемые теперь на полупроводниковых элементах, оставались конструктивно *дискретными*, т.е. состоящими из отдельных компонентов – транзисторов. Но уже первые полупроводниковые машины имели значительно меньшие габариты, меньшее тепловое излучение и были более надежны в работе, чем их электровакуумные предшественники.

В начале 60-х годов совершенствование вычислительных машин на базе полупроводников происходило по двум направлениям. Прежде всего создавались машины-гиганты, подобные ЭВМ фирм IBM, Burrougs и Honeywell, для которых по-прежнему требовались большие помещения с кондиционерами. Эти сложные машины могли обрабатывать большой объем данных и предназначались для коммерческих и научных расчетов. Однако они все еще оставались очень дорогими, и, чтобы окупить расходы, связанные с их приобретением, надо было эксплуатировать эти ЭВМ круглосуточно в течение многих недель. Для получения максимальной выгоды от использования этих дорогостоящих машин были разработаны два режима работы: *пакетный* и *разделения времени*. В пакетном режиме единовременно выполняется только одно задание, по окончании которого немедленно начинается выполнение следующего задания. В режиме разделения времени одновременно выполняется несколько заданий с поочередным переходом от выполнения части одного задания к некоторой части другого задания и т.д.

С появлением новых, сравнительно небольших фирм – изготовителей вычислительных машин связано другое направление в технологии их производства – создание малых ЭВМ, соизмеримых по размерам с рабочим столом. Являясь не столь мощными, как машины-гиганты, эти *мини-ЭВМ* значительно дешевле и предоставляют пользователю целый ряд возможностей больших машин.

Мини-ЭВМ быстро нашли свое место в лабораториях. Ученые по достоинству оценили эти машины, ориентированные на решение определенного класса задач. Вместо того чтобы выполнять задания различных типов на универсальной машине-гиганте, пользователь получил возможность переходить от одной машины, ориентированной на определенный класс задач, к другой машине, предназначенной для задач иного класса. Сама идея устройства, имеющего архитектуру ЭВМ и предоставляющего пользователю монополию выполнения одного задания, оказала существенное влияние на дальнейшее развитие вычислительной техники. Однако до тех пор, пока не появились дешевые мини-ЭВМ, мало кто мог позволить себе иметь вычислительную машину только для решения индивидуальных задач сравнительно узкого класса, ибо долгое время экономически неоправданным был такой режим работы, когда после кратковременного выполнения одного задания ЭВМ выключается и простаивает.

Совершенствование цифровых вычислительных машин происходило одновременно с развитием технологии производства полупроводниковых схем, при этом сокращалось разнообразие типов логических схем, используемых при конструировании

ЭВМ, и возрастал спрос на эти схемы. Указанные факторы явились предпосылкой создания в середине 60-х годов семейства *логических схем малого и среднего уровней интеграции (МИС и СИС)*. Технология производства *интегральных схем (ИС)* развивалась под влиянием двух тенденций: стремления к выпуску дешевой продукции и расширения ее функциональных возможностей за счет усложнения схемных решений.

Применение интегральных схем позволило увеличить вычислительные возможности мини-ЭВМ при тех же размерах. Настольные машины 60-х годов по мощности не уступали большим машинам 50-х годов, занимавшим отдельные комнаты. Новые в ту пору мини-ЭВМ размером с яшик письменного стола, стоимость которых составляла ~10 тыс. долл., соответствовали своим предшественникам – мини-ЭВМ с габаритами письменного стола и стоимостью ~100 тыс. долл. Начали даже появляться ЭВМ в форме одной монтажной панели, т.е. печатной платы, содержащей всю логическую часть, необходимую для выполнения функций процессора. Покупателю необходимо было только дополнить такой процессор источником питания и некоторыми внешними устройствами.

С середины 60-х годов происходил дальнейший прогресс в области технологии производства интегральных схем. К концу 60-х и началу 70-х годов получили широкое использование *интегральные схемы большого уровня интеграции (БИС)*, когда много разных функциональных возможностей реализовано в конструктивно единой схеме. Большинство первых БИС были ориентированы на выполнение определенных специальных функций, но некоторые типы этих схем имели универсальное назначение, например запоминающие устройства.

Появление электронных калькуляторов свидетельствовало о существенных достижениях в технологии производства БИС. Аппаратной базой первых калькуляторов являлось значительное количество (70–100) отдельных интегральных схем. Создание специальных БИС позволило строить калькулятор на пяти или шести модулях. К середине 70-х годов электронный калькулятор уже выполнялся на одной БИС.

Указанные достижения привели к следующему закономерному этапу в развитии вычислительной техники – реализации архитектуры ЭВМ на одной интегральной схеме, получившей название *микропроцессор*.

Благодаря появлению микропроцессоров стало возможным производство мощных калькуляторов и других средств вычислительной техники. Подобно мини-ЭВМ, ориентированной на выполнение сравнительно узкого класса задач, микропроцессор с архитектурой вычислительной машины может быть запрограммирован для выполнения одиночного задания. С появлением этих дешевых вычислительных средств стала практически доступной и широко распространенной такая продукция, как автоматические регуляторы температуры, печи высокочастотного нагрева, телефонные аппараты с памятью и т.п.

С начала 70-х годов основные усилия ученых и инженеров были направлены на совершенствование архитектуры микропроцессоров с целью увеличения их быстродействия и мощности. Первые микропроцессоры обрабатывали единовременно 4 бит данных, представленных в цифровой форме, т.е. использовались 4-разрядные слова. Скорость работы таких микропроцессоров была низкой, и они значительно уступали мини-ЭВМ. Однако вскоре появилось новое поколение микропроцессоров: на смену 4-разрядным пришли 8- и 16-разрядные микро-ЭВМ. Набор команд, выполняемых микропроцессором, расширился, а сами команды стали более сложными, с большими функциональными возможностями. Некоторые микропроцессоры по своим рабочим характеристикам сравнялись с мини-ЭВМ, а в отдельных случаях даже превзошли их.

Задания для самопроверки

Чтобы убедиться в правильности понимания изложенного выше материала, ответьте на следующие вопросы.

1. Как называется принцип организации обработки и перемещения данных внутри микропроцессора: а) пакетная обработка, б) архитектура, в) модульность, г) снабжение питанием?

2. Программа работы микропроцессора

хранится в памяти вместе с данными. Какой из перечисленных ниже объектов является единственным отличительным признаком между данными и командами программы: а) длина слова, б) тип слова, в) местоположение в памяти, г) факт выполнения или невыполнения микропроцессором определенных действий?

3. Мини-ЭВМ «принесли» с собой принцип построения процессора, ориентированного на выполнение узкого класса задач. а) Этот ли принцип положен в основу использования микропроцессора в современном телефонном аппарате с памятью или б) работа такого аппарата – это пример выполнения множества разных задач, в) является ли реализация такого принципа доказательством нереализуемости принципа запоминаемой программы или г) подтверждением реализуемости этого принципа лишь в будущем?

4. Что является причиной широкого использования интегральных схем при конструировании ЭВМ: а) реализация внутри машины одних и тех же логических функций, б) повышенные требования к быстродействию любой ЭВМ, в) отказ от более быстродействующих электровакуумных ламп, так как они излучают слишком много тепла, или г) совокупность всех перечисленных выше причин?

1.2. Общая характеристика микропроцессора

Термин «микропроцессор», конечно, несет определенную информацию об устройстве, названном этим именем. Микропроцессор, который иногда называют сокращенно МП, базируется на логических схемах того же типа, что и *центральный процессор* (ЦП) цифровой вычислительной машины. В обоих случаях для манипулирования данными и выполнения вычислений под управлением программы используются цифровые схемы. Иначе говоря, микропроцессор – это устройство *обработки данных*.

В отличие от стандартного ЦП цифровые логические схемы микропроцессора реализованы на одной или нескольких БИС, а так как последние тоже называют микросхемами, то становится понятным происхождение термина «микропроцессор».

Совершенно очевидно, что обработка данных – одна из главных функций микропроцессора, включающая как вычисления, так и манипулирование данными. Схемы, выполняющие вычисления, образуют так называемое *арифметическо-логическое устройство (АЛУ)*, в результате работы которого данные изменяют свои значения. К функциям, выполняемым АЛУ, относятся сложение (Add), вычитание (Subtract), И (AND), ИЛИ (OR), сравнение (Compare), положительное приращение (Increment) и отрицательное приращение (Decrement). Для выполнения этих операций АЛУ необходимы данные. Так, для выполнения сложения двух чисел эти числа должны быть заблаговременно размещены в нужном месте. Но АЛУ не осуществляет перемещение данных ни до, ни после выполнения операции. Оно лишь выполняет операции над данными, обнаружив их в определенном месте. АЛУ работает подобно слепому жонглеру, который может выполнять удивительные трюки только после того, как ему вручены объекты манипулирования. Слепой жонглер не может взять предмет без посторонней помощи. Подобно слепому жонглеру, АЛУ ожидает, пока данные не будут размещены в определенных местах. Но как АЛУ получает данные, подлежащие обработке?

В микропроцессоре, за пределами АЛУ, имеются другие схемы, манипулирующие данными, и в частности перемещающие их в места, доступные АЛУ. После того как АЛУ выполнило требуемые операции, эти схемы пересылают данные другим адресатам. Но если упомянутый жонглер не может сменить объекты манипулирования, то АЛУ выполняет операции, меняющие данные. Кто же информирует АЛУ, как обрабатывать данные, какие из возможных операций должны быть выполнены?

Управление системой – другая главная функция микропроцессора. Схемы управления позволяют декодировать (расшифровывать) и выполнять программу – набор команд для обработки данных. Схемы управления записывают *команды* (шаги программы) в память на хранение и извлекают их оттуда одну за другой. После из-

влечения команды из памяти микропроцессор декодирует ее. Схемы управления контролируют процесс выполнения декодированной команды.

Поскольку команды хранятся в памяти, вы можете менять их по своему желанию, изменяя при этом характер обработки данных. Команды, которые вы записываете в память, определяют будущую работу микропроцессора. Это очень важный момент для правильного понимания функционирования микропроцессора.

Подведем итоги рассмотренному выше. Микропроцессор выполняет две функции – обработку и управление. Обработка включает перемещение данных с одного места на другое и выполнение операций над ними, управление определяет, как обрабатывать данные. Работа микропроцессора состоит из следующих шагов: сначала извлекается команда, затем логическая схема управления ее декодирует, после этого осуществляется выполнение этой команды. Эти шаги называют циклом *«выборка-выполнение»*. Для каждой команды, хранимой в памяти, микропроцессор выполняет один такой цикл.

Помимо извлечения команд из памяти и их выполнения схемы управления выполняют ряд других важных функций, таких, как обмен информацией с внешними устройствами, подсоединяемыми к процессору. Каким бы мощным ни казался процессор, он ничего не может делать сам по себе. Микропроцессору нужна помощь со стороны других устройств. Необходимы схемы для хранения команд программы, а также для ввода данных в микропроцессор и вывода из него (схемы ввода-вывода). Для хранения данных требуется дополнительная память. Микропроцессор нуждается в электропитании от внешнего источника. Обратимся к простейшим играм, выполненным на базе микропроцессора. Для их реализации необходима панель с клавиатурой для ввода данных и устройство отображения информации (дисплей) того или иного вида для получения ответа. В качестве источника питания нужна батарея. И все это должно быть компактно размещено в некотором корпусе.

Настал момент провести четкую грань между такими понятиями, как микропроцессор и *микро-ЭВМ*. Микропроцессор является основой – «сердцем» многих устройств, однако отдельный микропроцессор не представляет собой законченное, самостоятельно функционирующее устройство[1]. Схемы управления микропроцессора могут управлять работой других составных частей вычислительной системы, создаваемой на основе микропроцессора.

Задания для самопроверки

5. Какие из перечисленных ниже схем не входят в состав микропроцессора: а) логические, б) выполняющие вычисления, в) запоминающие (память) или г) все названные?

6. Для какой из нижеуказанных целей предназначен цикл работы выборка-выполнение микропроцессора: а) выполнения логических функций, б) реализации шагов программы, в) арифметических вычислений или г) работы микропроцессора в целом?

7. Каковы функции АЛУ: а) выполнение операции сложения, б) перемещение данных, в) декодирование команд или г) реализация всех перечисленных операций?

8. Каким образом можно внести изменения в работу микропроцессора: а) изменяя команды в памяти, б) вводя новые данные, в) выводя данные или г) увеличивая размер памяти?

9. Для управления какими из следующих схем: а) памяти, б) ввода, в) вывода или г) всеми вместе предназначены управляющие сигналы, генерируемые микропроцессором?

10. Микропроцессор не является системой, изолированной от других устройств. Какие из перечисленных ниже устройств необходимы (как минимум) для его функционирования: а) память, б) устройство ввода, в) устройство вывода или г) все указанные устройства?

---

[1] Некоторые микро-ЭВМ, конструктивно выполненные на одном кристалле, содержат схемы ввода-вывода, запоминающее устройство для данных или программ, однако перечисленное не является частью того, что именуется микропроцессором.

1.3. Что такое микро-ЭВМ?

Часто термины «микропроцессор» и «микро-ЭВМ» применяются к одному и тому же объекту, однако они имеют различное значение. Как следует из предыдущего раздела, микропроцессор – это интегральная схема, предназначенная для обработки данных и управления. Что же касается микро-ЭВМ, то она представляет собой законченную вычислительную систему, центральной частью которой является микропроцессор. В систему, кроме того, входят память и схемы, реализующие операции ввода-вывода (рис. 1.2). Оформляемая конструктивно в виде прибора с передней панелью микро-ЭВМ состоит из следующих блоков (монтажных плат): микропроцессора, *памяти произвольного доступа* (RAM – Random Access Memory) и ввода-вывода, соединяющего микро-ЭВМ с видеотерминалом – *буквенно-цифровым дисплеем* на катодно-лучевой трубке. На экране дисплея, подобном экрану телевизора, высвечиваются буквы и цифры. Такой видеотерминал имеет панель с клавиатурой для ввода данных. Описываемая вычислительная система располагает также источником питания. На рис. 1.3 показана блок-схема подобной системы. Следует обратить внимание на то, что для преобразования микропроцессора в микро-ЭВМ требуется значительное количество дополнительных схем.

Все блоки микро-ЭВМ соединяются между собой *шиной*, содержащей большое число сигнальных линий. Английское название шины – Bus происходит от латинского omnibus, что означает «ко всем». Каждый блок использует один и тот же набор сигналов для «общения» с другими блоками. Конструктивно микро-ЭВМ – это отдельная плата, вставляемая в определенное место общей монтажной панели с расположенной на ней шиной, электрическая схема которой показана на рис. 1.4, *а*. Некоторые особенности конструкции монтажной панели демонстрирует рис. 1.4, *б*.

Блок микропроцессора содержит *таймер*-генератор тактовых сигналов, синхронизирующих во времени работу процессора, а также *постоянную память* (ROM – Read Only Memory), где хранятся команды программы, предназначенной для ввода других программ с накопителей на магнитной ленте или дисках. Блок микропроцессора, как и другие блоки микро-ЭВМ, использует четыре интегральные схемы в качестве интерфейса с шиной, для которой обычно требуются сигналы более высокого уровня тока, чем те, которые генерируют большинство слаботочных логических схем. Являясь хранилищем данных и программ, блок памяти произвольного доступа помимо четырех схем, подобных указанным выше, имеет дополнительные интегральные схемы. Блок ввода-вывода содержит УАПП – *универсальный асинхронный приемопередатчик* (UART – Universal Asynchronous Receiver-Transmitter) и таймер. Приемопередатчик преобразует данные из параллельного кода в последовательный, необходимый для связи с видеотерминалом. К четвертому самостоятельному блоку микро-ЭВМ следует отнести переднюю панель, содержащую, как и другие блоки, интегральные схемы – для интерфейса с шиной, а также логические схемы, управляющие работой *семисегментных световых индикаторов*. Источник питания – еще один блок микро-ЭВМ, соединяемый шиной со всеми другими блоками.

Как следует из описания системы в целом

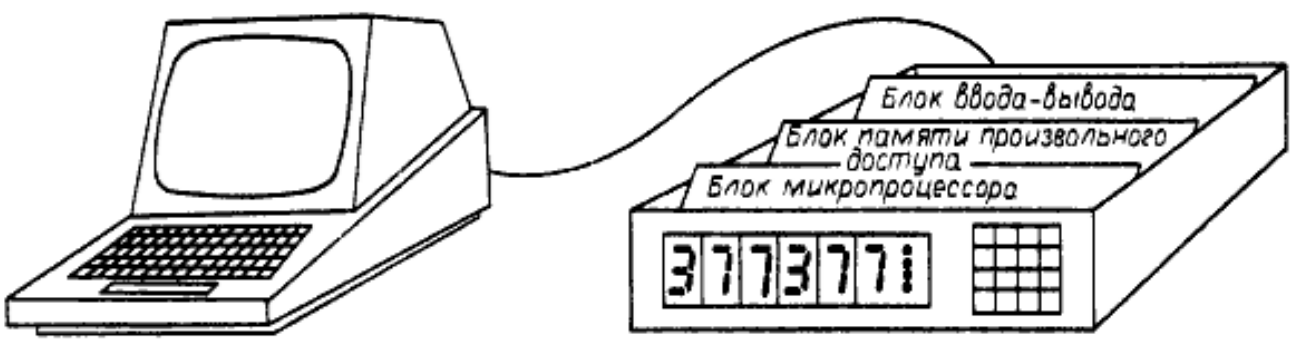


Рис. 1.2. Система с микро-ЭВМ, включающей микропроцессор, память произвольного доступа, блок ввода-вывода и источник питания.

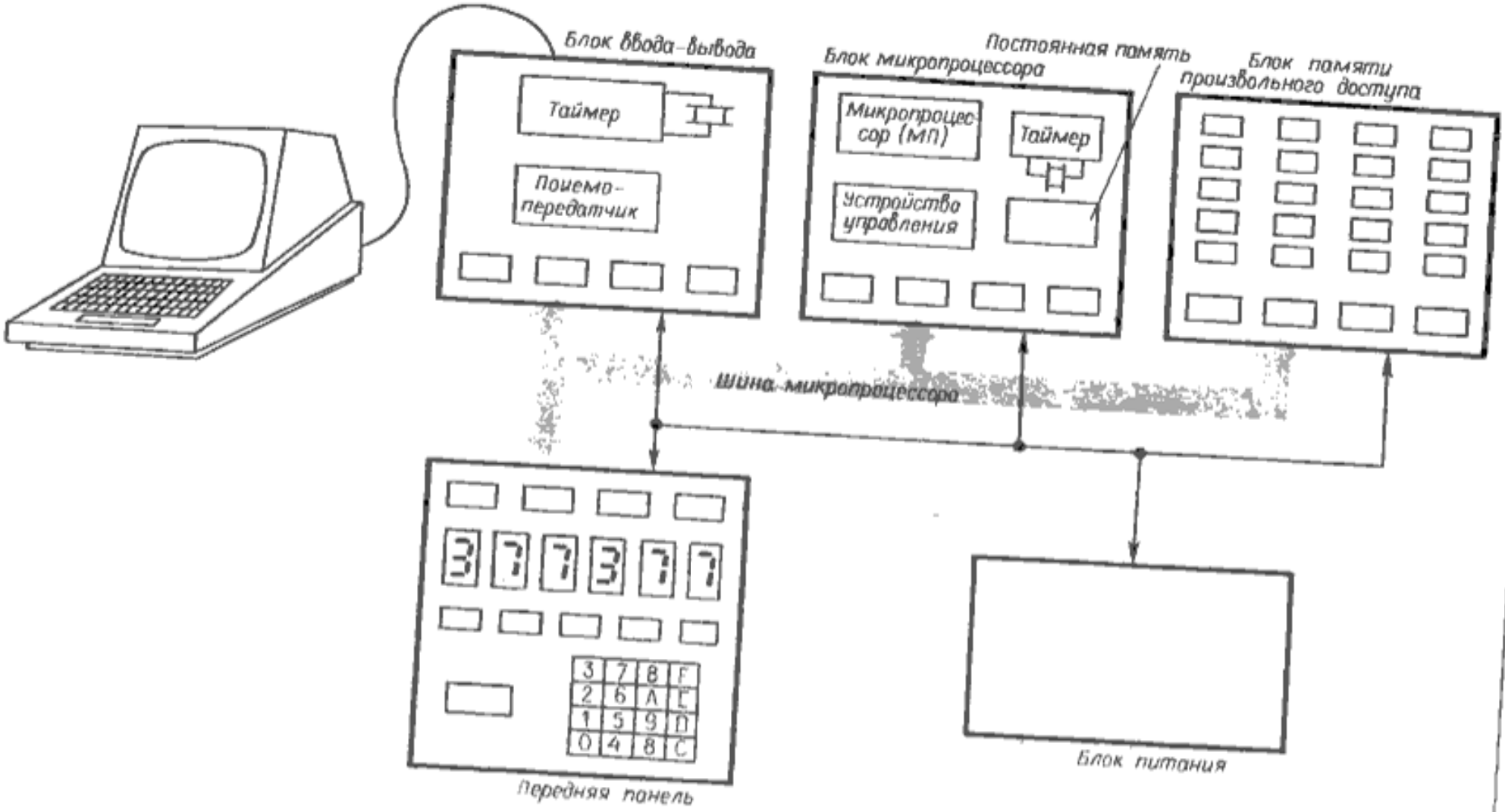
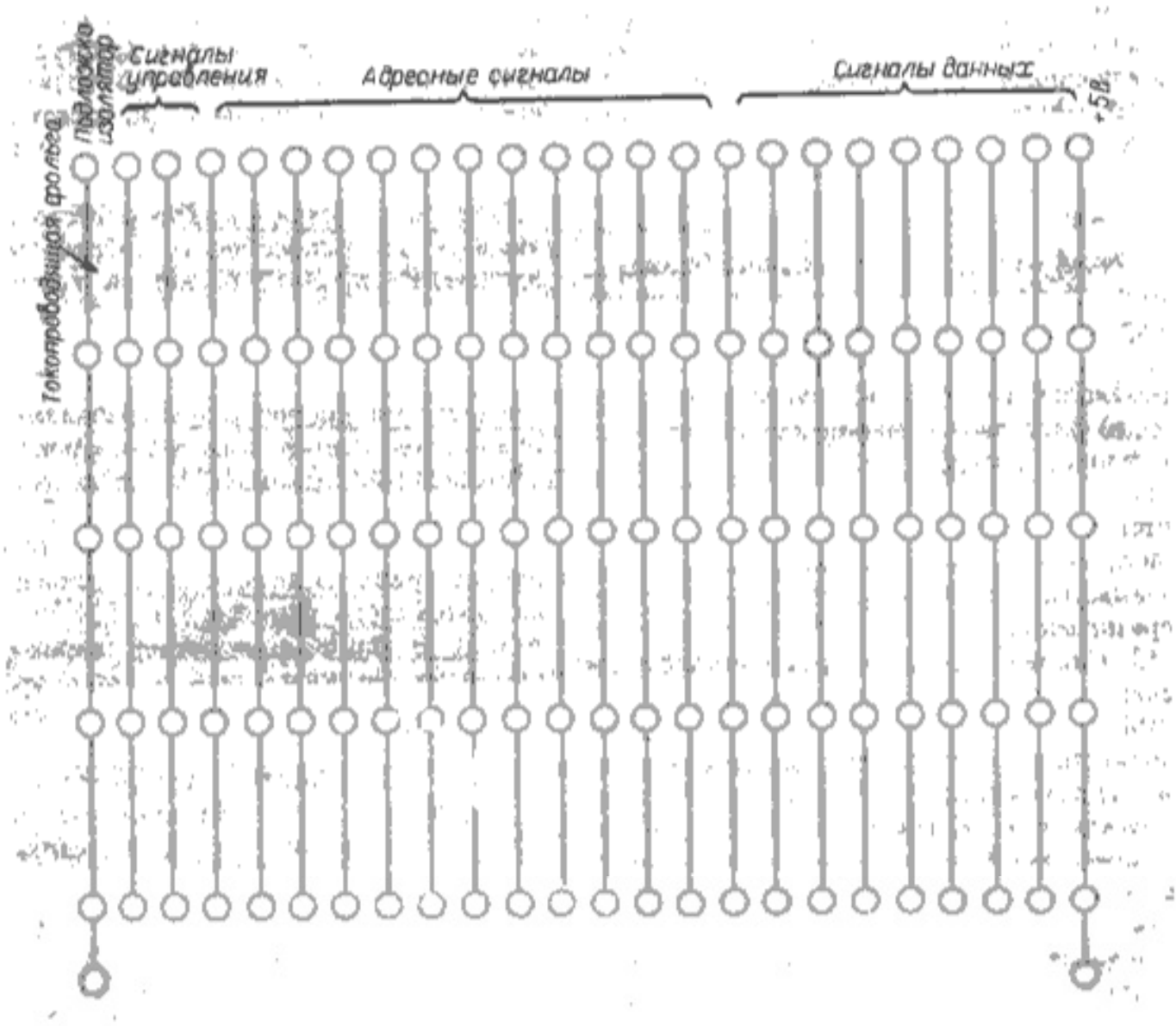
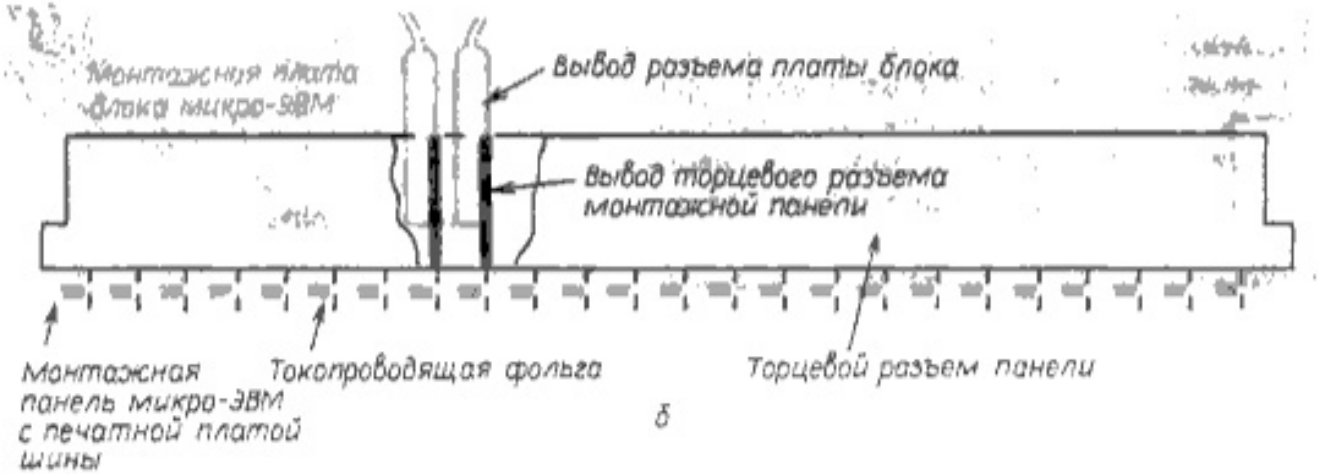


Рис. 1.3. Блок-схема микро-ЭВМ.



а



б

Рис. 1.4. *а*) Схематическое изображение электрических соединений типовой шины (торцевой разъем для монтажной платы каждого блока соединен токопроводящей фольгой печатной платы монтажной панели с такими же разъемами для других блоков микро-ЭВМ), *б*) вид монтажной панели сбоку (в разрезе).

(рис. 1.3), микропроцессор является хотя и малой, но жизненно важной частью микро-ЭВМ – законченной вычислительной системы. Обратим еще раз внимание читателя на тот факт, что вычислительная система как единое целое состоит по крайней мере из микропроцессора, памяти и блока ввода-вывода.

Задания для самопроверки

11. Какие из перечисленных ниже блоков необходимы для формирования простейшей микро-ЭВМ как законченной вычислительной системы: а) память (постоянная или произвольного доступа), б) микропроцессор, в) блок ввода-вывода или г) все перечисленные блоки?

12. В каком из блоков микро-ЭВМ хранятся ее команды: а) в памяти (постоянной или произвольного доступа), б) в блоке ввода-вывода, в) в микропроцессоре, г) во всех перечисленных блоках?

13. Каково назначение различных соединений сигнальных линий шины микро-ЭВМ: а) соединять между собой блоки микро-ЭВМ, б) устранять тепловое излучение, в) осуществлять связь с устройством, допускающим обмен данными лишь в последовательном коде, или г) использовать источник питания?

14. Для нормального функционирования отдельных схем в микропроцессор должны поступать сигналы синхронизации. Что является источником этих сигналов: а) постоянная память, б) память произвольного доступа, в) таймер или г) сам микропроцессор?

15. Микро-ЭВМ – это законченная вычислительная система. Говоря же о микропроцессоре, можно ли утверждать, что он является интегральной схемой: а) генерирующей сигналы для шины, б) управляющей блоком питания, в) обеспечивающей для системы обмен данными с внешними устройствами в последовательном коде, г) составной частью процессора ЭВМ?

1.4. Мощность микропроцессора

Почти все микропроцессоры изготовляются на кремниевых кристаллах. Длина стороны куба-кристалла равна 0,64 см. Такой кремниевый кристалл с содержащимися в нем электрическими схемами микропроцессора упаковывается в корпус так называемой интегральной схемы, имеющей от 16 до 64 выводов. Вот почему кристалл увидеть нельзя. Да в этом и нет необходимости, ибо его внешний вид мало что может сообщить нам о «мощности» заключенной в нем схемы микропроцессора.

Под «мощностью» микропроцессора понимают его способность обрабатывать данные. Ее принято оценивать тремя основными характеристиками: *длиной слова данных, количеством адресуемых слов памяти* и *скоростью выполнения команд*. Наиболее часто микропроцессоры сравнивают по длине слов данных. Каждый микропроцессор оперирует данными, представляемыми словами фиксированной длины, ибо в этом случае существенно упрощается построение процессора. В настоящее время типичными являются слова длиной 4, 8, 12 и 16 бит (разрядов). В ближайшее время широко доступными должны стать и микропроцессоры, оперирующие 32-битовыми словами. Рис. 1.5 иллюстрирует слова различной длины. Восьмибитовое слово используется столь широко, что даже получило специальное название *байт*. Благодаря распространенности байта даже 16-разрядные микропроцессоры часто имеют команды, обрабатывающие 16-битовые слова данных как пары байтов. На рис. 1.6 показано 16-битовое слово данных, составленное из двух 8-битовых слов: *старшего байта* (биты с 8-го по 15-й) и *младшего байта* (биты с 0-го по 7-й). Часто байт используется для оценки некоторого объема (пространства) в микро-ЭВМ. Например, иногда говорят, что программа микропроцессора занимает объем 4000 байт. В общем случае байт используют как некий «общий знаменатель» для измерения объема микропроцессора. Байт предпочитают слову данных, потому что размер последнего меняется в зависимости от типа процессоров, байт же всегда равен 8 бит. При одном и том же числе байтов 8-разрядный процессор содержит в два раза меньше слов, чем 4-разрядный, и в два раза больше, чем 16-разрядный. Например, 4000 байт 8-разрядного микропроцессора равно его 4000 словам; это же количество байтов 4-разрядного микропроцессора равно его

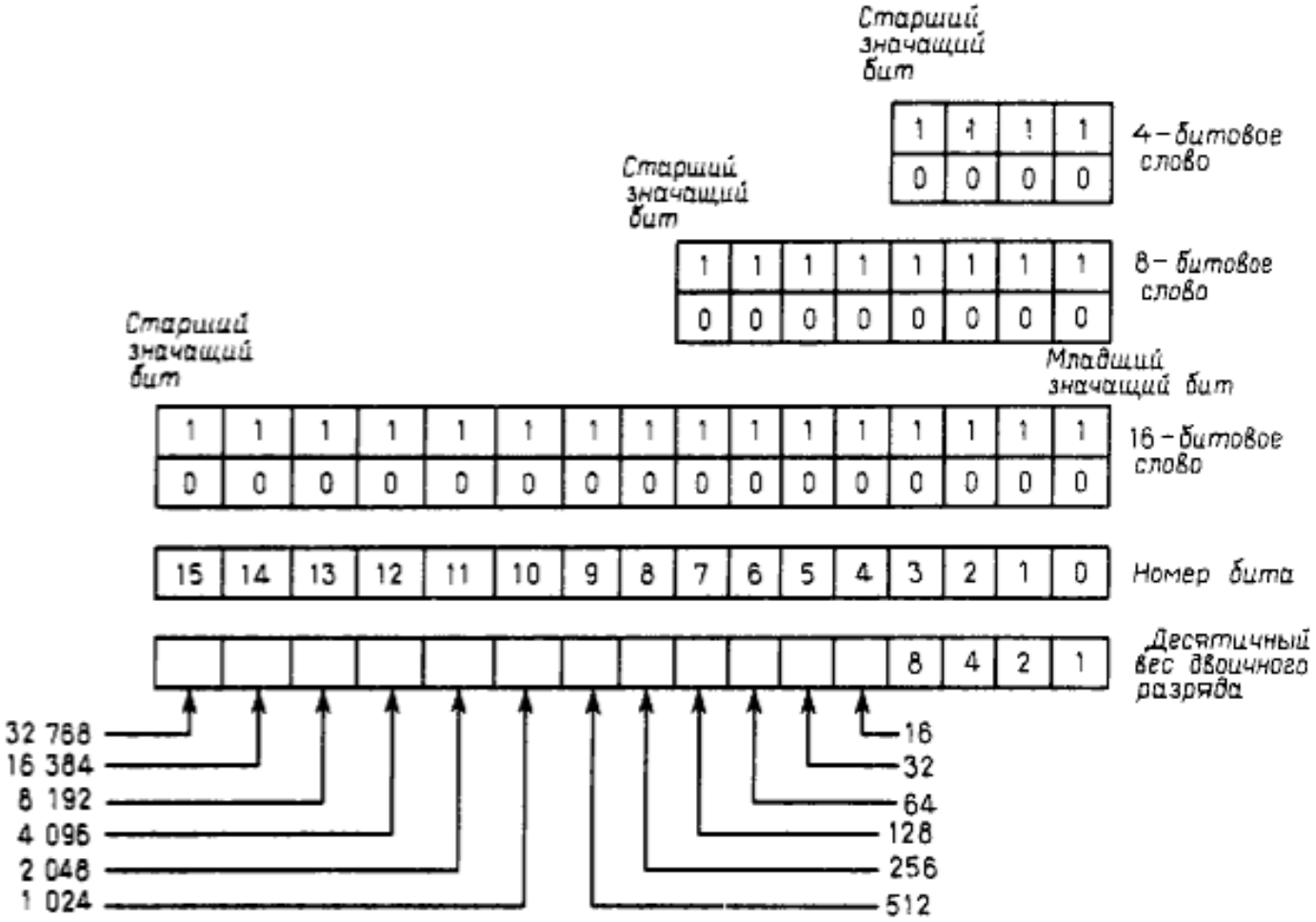


Рис. 1.5. 4-, 8- и 16-разрядные двоичные слова.

Позиции битов (двоичных разрядов) обычно нумеруются справа налево, начиная с нуля – позиции младшего значащего бита, при переходе справа налево «вес» бита увеличивается. Бит любого разряда может быть равен 1 или 0.

8000 словам, а для 16-разрядного микропроцессора – 2000 словам.

Первыми были разработаны 4-разрядные микропроцессоры, которые и сейчас еще находят применение. Четыре бита – это длина двоично-кодированного десятичного числа, т.е. числа в коде BCD (Binary-Coded Decimal). В калькуляторах, промышленных системах управления и некоторых других устройствах микропроцессору приходится иметь дело только с числами в коде BCD. В таких случаях использование 4-разрядного микропроцессора – это идеальное решение. Другой причиной продолжающегося производства и применения 4-разрядных микропроцессоров является их чрезвычайно низкая стоимость.

Восьмиразрядные микропроцессоры также широко распространены и недороги. Они появились вслед за 4-разрядными микропроцессорами, во-первых, потому, что слово длиной 8 бит – это два 4-битовых слова, во-вторых, поскольку 8-битовое слово позволяет «упаковать» в нем два числа в коде BCD, в-третьих, 8 бит позволяют представить символ (букву, цифру, знак) в коде ASCII (American Standard Code for Information Interchange – Стандратный американский код для обмена информацией), широко используемом при обработке информации.

Большинство первых 16-разрядных микропроцессоров выпускались как стандартные 16-разрядные мини-ЭВМ в виде БИС. Примером могут служить микропроцессоры LSI-11 фирмы Digital Equipment, являющиеся копией мини-ЭВМ PDP-11, микропроцессоры DGMN фирмы Data General

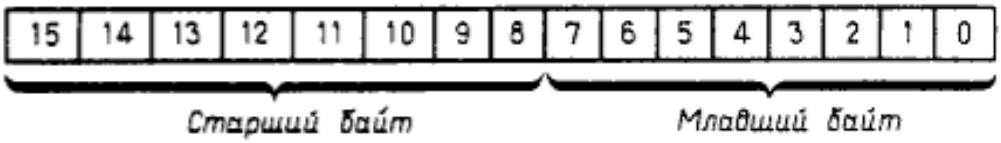


Рис. 1.6. Структура 16-битового двоичного слова.

MicroNova, родственные мини-ЭВМ Nova, микропроцессоры 9900 фирмы Texas Instruments, аналогичные мини-ЭВМ 990. Позже стали появляться 16-разрядные микропроцессоры, имеющие присущую только им архитектуру, которая не является копией архитектуры какой-либо мини-ЭВМ.

С каждым удваиванием длины слова микропроцессора последний становится более мощным. Увеличение длины слов вызвало совершенствование технологии БИС. Результатом этого явилось улучшение и других характеристик 16-разрядных микропроцессоров.

Другой «мерой» мощности микропроцессора является количество слов или байтов памяти, к которым он может адресоваться. И в этом случае длина слова данных играет важную роль. Длина слова данных в памяти – это, по существу, тот же параметр, что и длина слова данных, которыми оперирует микропроцессор. Так, 4-разрядный микропроцессор оперирует 4-битовыми словами данных, хранимыми в памяти.

Каждому слову в памяти присваивается номер его местоположения – так называемый *адрес*. Для извлечения слова из памяти ЭВМ обращается по соответствующему адресу. Адреса памяти начинаются с нуля и представляются в двоичной форме. ЭВМ разного типа имеют различные значения максимального адреса памяти. Чем больше значение максимального адреса памяти, тем больше вычислительная мощность микропроцессора. Табл. 1.1 демонстрирует способность адресации к памяти одного 4-битового слова: 4 бит позволяют получить доступ к 16 различным словам памяти. Таким образом, *диапазон адресации* 4-битового слова равен 16 словам. Диапазон адресации 8-битового слова равен 256 словам памяти, а 16-битового слова – 65 536 словам. Конечно, возможность адресации к данным, находящимся в памяти, большинства процессоров не ограничена единственным словом, поэтому и диапазон адресации не ограничен длиной слова данных. Адреса памяти могут иметь длину 22 бит и более. Некоторые микропроцессоры обращаются к миллионам слов памяти.

В табл. 1.2 показаны диапазоны адресации для некоторых широко распространенных микропроцессоров, оперирующих

Таблица 1.1. **Память, состоящая из 16 слов, которая адресуется 4-битовым словом**

Двоичный адрес	Содержимое памяти
1111	15-е слово данных
1110	14-е слово данных
1101	13-е слово данных
1100	12-е слово данных
1011	11-е слово данных
1010	10-е слово данных
1001	9-е слово данных
1000	8-е слово данных
0111	7-е слово данных
0110	6-е слово данных
0101	5-е слово данных
0100	4-е слово данных
0011	3-е слово данных
0010	2-е слово данных
0001	1-е слово данных
0000	0-е слово данных

словами длиной 4, 8 и 16 бит. Для 4-разрядных микропроцессоров обычными являются диапазоны адресации, равные 4096 и 8192 словам памяти. Для 8-разрядных микропроцессоров зачастую этот диапазон составляет 65 536 слов. Что же касается 16-разрядных микропроцессоров, то значения диапазона могут простираться от 32 768 до 4 194 304 слов.

При работе с микропроцессорами часто приходится ссылаться на подобные большие числа. Поскольку эти числа не круглые, их нелегко запомнить, а их произношение затруднительно. Поэтому предложены сокращенные обозначения этих чисел, упрощающие манипулирование ими. В табл. 1.2 эти сокращения указаны в круглых скобках. Так, диапазон адресации, равный 4096 словам, есть диапазон 4К. Буква К происходит от приставки «кило», заимствованной из греческого языка и означающей «тысяча». В данном случае – это условное обозначение величины 1000. Выражение «4К» означает число 4096, округленное до ближайшей тысячи. Аналогично формируются краткие формы записи других размеров памяти. Когда эти размеры оказываются чрезвычайно большими, используется буква М, соответствующая приставке «мега», т. е. миллион.

Для 8-разрядных микропроцессоров типичной является память размером 65 Кбайт. Более современные 16-разрядные микропроцессоры оперируют адресными

Таблица 1.2. **Размеры слова и адресного пространства (диапазона адресации) памяти некоторых типичных микропроцессоров**

Длина слова данных	4 бит	8 бит	16 бит
Диапазон адресации	4096 (4K)		
	8192 (8K)		
		65 536 (65K)	
			32 768 (32K)
			65 536 (65K)
			1 048 576 (1M)
			2 097 152 (2M)
			4 194 304 (4M)

пространствами, простирающимися до нескольких мегабайт.

Говоря о размере памяти микро-ЭВМ, следует помнить, что этот параметр может быть выражен в байтах или словах. В данной книге в дальнейшем для этой цели будут использоваться только байты. А поскольку 1 байт равен 8 бит, то память в 65 536 слов 8-разрядной микро-ЭВМ имеет тот же размер, что и память в 32 768 слов 16-разрядной микро-ЭВМ. В обоих случаях размер памяти равен 65 536 байт. Если же это сравнение непонятно, можно сравнить размеры памяти в битах, умножая величину, выраженную в словах памяти, на длину слова в битах:

65 536 слов × 8 бит/слово = 524 288 бит и
32 758 слов × 16 бит/слово = 524 288 бит.

Третьей характеристикой микропроцессора, используемой для измерения его мощности, является скорость, с которой он выполняет команды. Она оценивается продолжительностью во времени цикла «выборка-выполнение» для одного шага программы. Скорость одних микропроцессоров в 20–100 раз превосходит скорость других. «Медленные» микропроцессоры используют генератор тактовых импульсов частотой в несколько сотен килогерц. Для выполнения одной команды им требуется от 10 до 20 мкс. Тактовая частота быстродействующих микропроцессоров достигает 5–10 МГц, что позволяет выполнить команду за несколько десятых долей микросекунды. Поскольку скорость работы микропроцессора связана с максимальной частотой генератора тактовых импульсов, иногда микропроцессоры сравнивают по значению этой частоты. Однако больший смысл имеет сравнение продолжительности выполнения одной и той же операции на разных процессорах. Для упрощения подобных сравнений создаются специальные короткие программы так называемого *контрольного прогона*. Быстродействие различных микропроцессоров оценивается путем сравнения времени выполнения подобной программы.

Задания для самопроверки

16. Чем принято измерять мощность микропроцессора: а) размером (объемом) кристалла, б) длиной слова, в) количеством выводов или г) совокупностью перечисленных характеристик?

17. Как принято называть длину слова, состоящего из 8 бит: а) байтом, б) ЦП, в) МП или г) адресом?

18. Двенадцатибитовые слова используются в семействе мини-ЭВМ PDP-8 и в некоторых других микропроцессорах. Какова выраженная в битах емкость памяти машины PDP-8: а) 32 768, б) 262 144, в) 384 000 или г) 393 216, если в словах она равна 32K?

19. Многие 8-разрядные микропроцессоры могут адресоваться к 65 536 областям памяти, размер которой обозначается обыч-

но 65 К. Иногда допустимо использование записи 64К для обозначения объема той же самой памяти. В обоих случаях, т.е. и 65К, и 64К, означают величину 65 536. Почему допустимы оба обозначения?

20. Две идентичные микро-ЭВМ имеют различные значения частоты генератора тактовых сигналов: для машины А эта величина равна 5 МГц, для машины В период следования тактовых импульсов равен 1 мкс. Какая машина выполняет команды быстрее и почему?

Упражнения

1.1. Архитектура какого устройства является прообразом основных архитектурных решений микропроцессора: калькулятора на БИС, программируемой с пульта ЭВМ, цифровой ЭВМ общего назначения или электронно-лучевой трубки?

1.2. Какие достижения технологии производства полупроводников сделали возможным изготовление микропроцессоров: создание монтажной платы блока микропроцессора, создание БИС, разработка светоизлучающих диодов или появление германиевых транзисторов?

1.3. Работа микропроцессора основана на принципе запоминаемой программы. В каком из блоков микро-ЭВМ – ввода-вывода в последовательном коде, микропроцессора, питания или памяти – хранятся команды вместе с данными?

1.4. Использование архитектуры ЭВМ в сочетании с микропроцессорной технологией позволило создавать вычислительные системы для решения простых задач, для которых применение ЭВМ общего назначения неоправданно. В чем состоит преимущество микропроцессоров в этом случае: в быстродействии, чрезвычайно низкой стоимости, мощности или в сочетании этих качеств?

1.5. Что означает сокращение МП? Функциям какого блока ЭВМ общего назначения подобны функции МП?

1.6. Микропроцессор осуществляет ряд функций по манипулированию данными. Каково назначение этих функций: выполнять логическую операцию И, перемещать данные между отдельными частями микро-ЭВМ, производить вычисления над данными или обеспечивать бóльшую скорость работы по сравнению с выполнением только вычислений?

1.7. Для выполнения одной команды микропроцессор осуществляет цикл «выборка-выполнение». Где реализуется часть этого цикла, именуемая «выполнение»: в АЛУ, блоке ввода-вывода, памяти произвольного доступа или в постоянной памяти?

1.8. Микропроцессор – это «сердце» микро-ЭВМ. Какие еще блоки должна содержать эта вычислительная машина: памяти, ввода, вывода или все перечисленные блоки?

1.9. Какое из следующих ниже утверждений в полной мере применимо к микропроцессору:

а) всегда работает со скоростью ~1 МГц,

б) располагает памятью размером не менее 16К байт,

в) всегда представляет собой один кристалл,

г) никогда не функционирует самостоятельно?

1.10. Каково назначение шины микро-ЭВМ:

а) предоставлять промышленности возможность выпускать стандартные изделия,

б) обеспечивать коммуникации между отдельными частями микро-ЭВМ через систему сигнальных линий,

в) обеспечивать для микро-ЭВМ стандартное механическое соединение,

г) гарантировать надежную передачу сигналов частотой 3,58 МГц и выше?

1.11. Часто шина микро-ЭВМ – это набор параллельных проводников, соединяющих между собой разъемы, в которые вставляются монтажные платы отдельных блоков машины. Как вы думаете, почему выбрана именно такая конструкция?

1.12. Длина слова данных, равная 8 бит, широко распространена. Обычно для адресации той или иной области памяти используются два слова. Чему равно в таком случае общее количество адресуемых областей памяти: 16К, 32К, 65К или 128К?

1.13. Какова длина слова, используемая в микропроцессорах: 4, 8, 16 бит или употребляются слова всех указанных размеров?

1.14. Как принято называть слово длиной 8 бит: широко используемым, байтом, МП или ЦП?

1.15. Чему равно количество областей памяти, адресуемых 16-разрядным микропроцессором: 65 536, 64К, некоторой переменной величине или нескольким миллионам?

1.16. Что определяет время цикла «выборка-выполнение»: диапазон адресации, частота генератора тактовых импульсов, размер шины или все перечисленные характеристики?

Ответы на вопросы заданий для самопроверки

1. б. 2. в. 3. б. 4. а. 5. в. 6. б. 7. а. 8. а. 9. г. 10. г. 11. г. 12. а. 13. а. 14. в. 15. в. 16. б. 17. а. 18. г. 19. Поскольку $64 = 2^6$, то 64 000 иногда считают ближайшим круглым числом к 65 536. 20. Машина А, потому что период следования импульсов генератора тактовых сигналов 5 МГц равен 200 нс, что в пять раз короче периода, равного 1 мкс, или 1000 нс.

Глава 2. Десятичная и двоичная системы счисления

Эта глава начинается обзором двоичной и десятичной систем счисления, последняя из которых нам хорошо знакома – мы с ней встречаемся ежедневно. Так, при изучении электронных схем приходится выполнять много расчетов, пользуясь десятичной системой. Однако для овладения цифровой техникой необходимо иметь полное представление о двоичной системе. Знание обеих систем счисления важно для понимания функционирования цифровых вычислительных устройств, и в частности микропроцессоров. По этим причинам далее рассматриваются преобразование чисел из десятичной системы в двоичную и обратное преобразование. Здесь же описываются два широко используемых способа краткой записи больших двоичных чисел – с помощью восьмеричной и шестнадцатеричной систем счисления. Восьмеричной записью пользуются производители микропроцессоров, а шестнадцатеричная запись находит применение в других областях вычислительной техники.

2.1. Десятичная система счисления

Это наиболее широко распространенная система счисления, использующая десять различных символов для представления любой величины. Поскольку символов десять (0, 1, 2, 3, 4, 5, 6, 7, 8 и 9), то говорят, что это *система с основанием 10*. Заметим, что 0 служит одним из символов десятичной системы счисления. Это очень важно помнить и объясняет тот факт, что ЭВМ, а следовательно, и микропроцессор начинают счет с нуля.

Что необходимо предпринять для подсчета объектов, общее количество которых превышает 10? Для этого слева от исходной позиции, занимаемой цифрой, нужно добавить еще одну или несколько цифр. Можно сказать, что следующее слева число указывает, сколько раз была полностью использована позиция справа. Рассмотрим, например, число 23, выражающее количество некоторых объектов. Процедуру формирования записи этого числа можно описать следующими действиями:

а) Отыскиваем достаточное количество объектов для использования всех символов (от 0 до 9) в крайней справа позиции (при этом содержимое соседней позиции слева равно 0),

б) вторично отыскиваем достаточное количество объектов для использования всех символов (от 0 до 9) в крайней справа позиции (при этом содержимое соседней позиции слева полагаем равным 1),

в) в третий раз отыскиваем достаточное количество объектов для использования всех символов (от 0 до 9) в крайней справа позиции (отмечаем этот факт записью символа 2 в соседнюю позицию слева),

г) четвертое повторение описанных выше действий позволяет использовать в крайней справа позиции символ 3, поскольку на этот раз обнаружены лишь четыре объекта (использованы символы 0, 1, 2 и 3).

Таблица 2.1

Количество объектов	Позиции цифр десятичного числа		Объекты
	вторая	первая	
0	0	0	Отсутствуют
1	0	1	.
2	0	2	..
3	0	3	...
4	0	4	....
5	0	5	.....
6	0	6	.....
7	0	7	.....
8	0	8	.....
9	0	9	.....
10	1	0	.....
11	1	1	.....
12	1	2	.....
13	1	3	.....
14	1	4	.....
15	1	5	.....
16	1	6	.....
17	1	7	.....
18	1	8	.....
19	1	9	.....
20	2	0	.....
21	2	1	.....
22	2	2	.....
23	2	3	.....

Описанный способ счета символически изображен в табл. 2.1, где каждое число обозначает соответствующее количество объектов. Обычно *незначащие* нули, т. е. нули, расположенные слева от первой значащей цифры (см. табл. 2.1), принято опускать.

Каждую позицию цифры в числе принято оценивать «весом» – *показателем степени* числа 10. В первой справа позиции размещены единицы, в соседней с ней второй позиции – десятки, в третьей – сотни, в четвертой – тысячи и т. д. Присвоение десятичного веса каждой позиции десятичного числа 6321 можно представить следующим образом:

100000	10000	1000	100	10	1	1/10	1/100
0	0	6	3	2	1.	0	0

т. е. вес крайней справа позиции равен 10^0, соседней с ней – 10^1, следующей – 10^2, четвертой – 10^3 и т. д. Иначе говоря, вес позиции равен основанию системы счисления, возведенному в номер этой позиции. Для того же самого числа 6321 веса позиций (разрядов) можно записать по-другому:

10^3	10^2	10^1	10^0
6	3	2	1

Тогда можно определить величину, представленную этими цифрами:

$$6 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 = 6321.$$

В качестве еще одного примера рассмотрим число 1 260 523, веса разрядов которого определим, пользуясь следующим представлением:

10^6	10^5	10^4	10^3	10^2	10^1	10^0
1	2	6	0	5	2	3

Тогда

$$1 \times 10^6 + 2 \times 10^5 + 6 \times 10^4 + 0 \times 10^3 + 5 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 = 1\,260\,523.$$

Дробная часть десятичного числа находится справа от десятичной точки, используемой для отделения целой части числа от дробной. Каждая позиция справа от деся-

тичной точки имеет свой вес. Ближайшая к точке позиция имеет вес, равный 1/10, следующая за ней – вес, равный 1/100 и т. д. То есть вес позиции дробной части числа равен основанию системы счисления, возведенному в отрицательное значение номера позиции; счет номеров ведется от десятичной точки слева направо. Так, вес первой позиции справа от десятичной точки равен 10^{-1}. Например:

10^3	10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}
6	3	2	1.	5	6	4

Следовательно, результат равен

$$6 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 + 5 \times 10^{-1} + 6 \times 10^{-2} + 4 \times 10^{-3} = 6321.564.$$

Задания для самопроверки

Чтобы убедиться в правильном понимании изложенного выше материала, ответьте на следующие вопросы.

1. С какой стороны от десятичной точки [а) справа, б) слева, в) напротив или г) вопрос некорректен] располагаются позиции цифр десятичного числа, имеющие вес, равный числу 10, возведенному в положительную степень?

2. Какое из нижеперечисленных действий необходимо выполнить для определения «вклада» данного разряда числа в общую величину последнего: а) сложить номер разряда (позиции) со значением содержащейся в нем цифры, б) разделить номер разряда на эту цифру, в) перемножить указанные величины, г) вычесть номер разряда из расположенной в нем цифры?

3. Чему равно значение следующего выражения:

$$2 \times 10^4 + 7 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 9 \times 10^0?$$

4. Веса позиций некоторых цифр десятичного числа равны числу 10, возведенному в отрицательные значения номеров этих позиций. Какая часть десятичного числа занимает эти позиции: а) половина, б) целая, в) все части или г) дробная?

2.2. Двоичная система счисления

Двоичная система счисления проще десятичной. В ней используются только два символа, что хорошо согласуется с техническими характеристиками цифровых схем, имеющих лишь два устойчивых состояния. Как правило, в качестве символов в двоичной системе служат 0 и 1. Иногда используются другие обозначения: термины «включено – выключено», «низкий – высокий уровень», «знак – пробел». Однако в любом случае речь идет о двух возможных состояниях.

При сравнении записи десятичных и двоичных чисел выясняется, что последние занимают значительно больше позиций, поскольку для их представления используются лишь два символа. Что же касается десятичной системы, то она «оперирует» не двумя, а десятью различными символами.

В двоичной системе счисления, так же как и в десятичной, каждой позиции (разряду) присвоен определенный вес. Но если в десятичной системе вес равен числу 10 в некоторой степени, то в двоичной системе вместо числа 10 используется число 2. Веса первых 13 позиций (разрядов) цифр двоичного числа имеют следующие значения:

4096	2048	1024	512	256	128	64	32	16	8	4	2	1
2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

В двоичной системе счисления даже сравнительно небольшие числа занимают много позиций. Например:

$$101101_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 45_{10},$$

т. е. двоичное число 101101 имеет ту же величину, что и десятичное число 45. Для удобства идентификации записи двоичных и десятичных чисел в виде нижнего индекса числа записывают 2 и 10 соответственно. Так, например, 101_2 – двоичное число, а 101_{10} – десятичное, причем они выражают разные величины.

Как и в десятичной системе, в двоичной системе для отделения дробной части от целой используется точка (*двоичная точка*). Каждая позиция справа от этой точки имеет свой вес – вес разряда дробной части числа.

Значение веса в этом случае равно основанию двоичной системы, возведенному в отрицательную степень. Такие веса – это дроби 1/2, 1/4, 1/8, 1/16, 1/32 и т.д., которые могут быть записаны как 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, 2^{-5} и т.п. Выразим эти веса через десятичные дроби: $2^{-1} = 0.5$, $2^{-2} = 0.25$, $2^{-3} = 0.125$, $2^{-4} = 0.0625$, $2^{-5} = 0.03125$, $2^{-6} = 0.015625$ и т.д.

При записи числа в десятичной системе каждая позиция занята *десятичной цифрой*. Аналогично при записи двоичного числа каждая позиция занята *двоичной цифрой*, называемой *битом*. Говоря о двоичных числах, часто пользуются понятиями *наименьший значащий бит* (*самый младший двоичный разряд*) и *наибольший значащий бит* (*самый старший двоичный разряд*) по аналогии с наибольшей и наименьшей значащими цифрами десятичного числа. Наименьший значащий бит имеет наименьший вес, а наибольший значащий бит соответственно наибольший. Обычно двоичное число записывается так, что наибольший значащий бит является крайним слева.

Задания для самопроверки

5. Чем является цифра 2 в десятичном числе 2364: а) наименьшей значащей десятичной цифрой, б) наибольшей значащей десятичной цифрой, в) наименьшим значащим битом или г) наибольшим значащим битом?

6. Является ли двоичным число 1210110? Если является, то почему?

7. Что означает термин «наименьший значащий бит»?

8. Что означает понятие «бит»?

9. Что является основанием двоичной системы счисления: а) 10_{10}, б) 1_2, в) 2_{10} или г) 0_{10}?

10. Какое из двух чисел больше: 1111_2 или 11_{10}? Обоснуйте ответ.

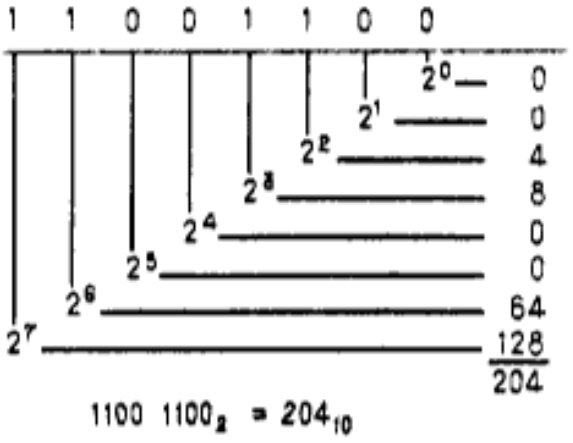
11. Какому из следующих далее чисел равно число 2^{-3}: а) 1/16, б) 1/8, в) 1/4 или г) 1/2?

2.3. Преобразование двоичных чисел в десятичные

При работе с микро-ЭВМ часто бывает необходимо заменить двоичные числа их десятичными эквивалентами.

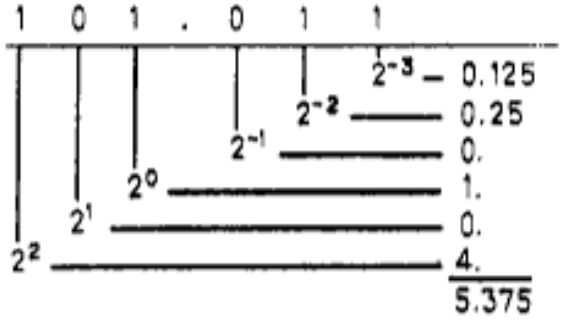
Процедура преобразования двоичного числа в десятичное проста: необходимо сложить десятичные веса всех разрядов двоичного числа, в которых содержатся единицы. Продемонстрируем это на следующих примерах.

Пример 1. Преобразование целого двоичного числа 1100 1100 в десятичное:



$1100\ 1100_2 = 204_{10}$

Пример 2. Преобразование вещественного двоичного числа 101.011 в десятичное:



$101.011_2 = 5.375_{10}$

Задания для самопроверки

12. Определите десятичные значения следующих двоичных чисел:

а) 10110110
б) 01010000
в) 00011110
г) 10000000
д) 11111111
е) 10000001
ж) 01111110
з) 00111111
и) 0.0101
к) 100.011

2.4. Преобразование десятичных чисел в двоичные

Иногда бывает удобно для пользователя, чтобы микро-ЭВМ имела дело с десятичными числами. Для этого необходимо ввести такие числа в машину, преобразовать их в двоичные эквиваленты, побудить микропроцессор обработать двоичные числа, а затем полученные двоичные числа – результат преобразовать в десятичные числа. Следовательно, нужно владеть приемами преобразования десятичных чисел в двоичные. Во-первых, необходимо уметь выполнять требуемые операции вручную, пока осуществляется программирование работы микропроцессора. Это позволит нам ввести числовые константы в программу вычислений и провести арифметические расчеты. Во-вторых, мы должны быть способны использовать микропроцессор для преобразования десятичных чисел в двоичные. Только тогда пользователь может вводить десятичные числа, которые микропроцессор будет преобразовывать в двоичные. Освоив технику выполнения первой процедуры, легко разобраться и во второй, реализуемой программными средствами.

Процедура преобразования целых десятичных чисел в двоичные – это частный случай процедуры перевода чисел из одной системы счисления в другую. Предположим, что необходимо преобразовать десятичное число 10 в двоичное. Для этого проделаем следующее.

1. Разделим подлежащее преобразованию число на основание системы счисления, в которой число должно быть представлено. В данном случае 10 следует поделить на 2. При делении на 2 остаток может быть равен 1 или 0. Значение остатка присваивается младшему значащему разряду (МЗР) искомого числа. Для рассматриваемого примера частное равно 5, а остаток – нулю, т. е. 1-й разряд равен нулю.

$$\begin{array}{r} 5 \\ 2\overline{)10} \\ \underline{10} \\ 0 \end{array} \quad \text{МЗР} = 0$$

2. Результат деления на первом шаге необходимо разделить еще раз на 2. Остаток (0 или 1) используется в качестве значения следующего по значимости разряда. В данном случае частное от деления 5 на 2 равно 2, а остаток, т. е. значение 2-го разряда, равно 1.

$$\begin{array}{r} 2 \\ 2\overline{)5} \\ \underline{4} \\ 1 \end{array} \quad 2\text{-й разряд} = 1$$

3. Результат деления на предыдущем шаге необходимо разделить на 2, а значение остатка присвоить очередному разряду. В данном случае частное равно 1, а остаток равен нулю, т. е. 3-й разряд равен нулю.

$$\begin{array}{r} 1 \\ 2\overline{)2} \\ \underline{2} \\ 0 \end{array} \quad 3\text{-й разряд} = 0$$

4. Шаги описанной процедуры повторяются до тех пор, пока частное, полученное в результате очередной операции деления, не станет равным нулю. Тогда остаток от последнего деления используется в качестве значения старшего значащего разряда (СЗР). В данном случае частное от деления 1 на 2 составляет нуль, а остаток равен 1, поэтому значение 4-го разряда равно 1.

$$\begin{array}{r} 0 \\ 2\overline{)1} \\ \underline{0} \\ 1 \end{array} \quad \text{СЗР} = 1$$

Итак, получено целое двоичное число 1010. Рассмотрим еще два примера преобразования десятичных чисел в двоичные.

Пример 1. Преобразование десятичного числа 57_{10} в двоичное число:

Шаг	Деление	Частное	Остаток
1	57/2	28	1 (МЗР)
2	28/2	14	0
3	14/2	7	0
4	7/2	3	1
5	3/2	1	1
6	1/2	0	1 (СЗР)

Результат: $57_{10} = 111001_2$

Пример 2. Преобразование десятичного числа 134_{10} в двоичное число:

Шаг	Деление	Частное	Остаток
1	134/2	67	0 (МЗР)
2	67/2	33	1
3	33/2	16	1
4	16/2	8	0
5	8/2	4	0
6	4/2	2	0
7	2/2	1	0
8	1/2	0	1 (СЗР)

Результат: $134_{10} = 10000110_2$

Изложенная процедура применима к преобразованию целых (или целой части) десятичных чисел в двоичные. Для дробных чисел (или дробных частей вещественных чисел) требуется отдельная, хотя и похожая, процедура. Если преобразование выполнено отдельно для целой и дробной частей числа, то результат получают путем записи двоичных эквивалентов этих частей соответственно слева и справа от двоичной точки.

Процедуру преобразования десятичной дроби в двоичную рассмотрим на примере преобразования числа 0.375.

1. Преобразование осуществляется умножением дроби на основание системы счисления, в которой дробь должна быть представлена. В данном случае умножаем на 2: $2 \times 0.375 = 0.75$.

2. Если результат умножения меньше 1, то старшему значащему разряду присваивается значение 0; если больше 1, то присваивается 1. Поскольку $0.75 < 1$, то СЗР $= 0$.

3. Результат предыдущей операции умножения опять умножается на 2. Заметим, что если бы результат предыдущей операции умножения был больше 1, то в данной операции умножения участвовала лишь его дробная часть. В данном случае $2 \times 0.75 = 1.5$.

4. Если полученный результат меньше 1, то следующему по значимости (ближайшему справа) разряду присваивается значение 0; если равен или больше 1, то присваивается 1. В рассматриваемом примере $1.5 > 1$, поэтому значение разряда 2 равно 1.

5. Шаги описанной процедуры повторяются до тех пор, пока либо результат умножения не будет точно равен 1, либо не будет достигнута требуемая точность. В данном случае после выполнения очередного шага результат равен $(2 \times 0.5 = 1.0)$. Поэтому очередному разряду, являющемуся младшим значащим разрядом, присваивается значение 1.

Следовательно, получена двоичная дробь 0.011.

Следует отметить, что не всегда путем повторения операций умножения можно достичь результата умножения, точно равного 1. В таком случае процесс повторения останавливают по достижении необходимой точности, а целую часть результата последней операции умножения используют в качестве значения младшего значащего разряда.

Рассмотрим еще два примера преобразования десятичных дробей в двоичные.

Пример 1. Преобразование десятичного числа 0.34375_{10} в двоичное:

Умножение	Результат в целочисленной форме
$2 \times 0.34375 = 0.6875$	0 (СЗР)
$2 \times 0.6875 = 1.375$	1
$2 \times 0.375 = 0.75$	0
$2 \times 0.75 = 1.5$	1
$2 \times 0.5 = 1.0$	1
$2 \times 0 = 0$	0 (МЗР)

Результат: 0.01011_2

Пример 2. Преобразование десятичного числа 0.3_{10} в двоичное:

Умножение	Результат в целочисленной форме
$2 \times 0.3 = 0.6$	0
$2 \times 0.6 = 1.2$	1
$2 \times 0.2 = 0.4$	0
$2 \times 0.4 = 0.8$	0
$2 \times 0.8 = 1.6$	1
$2 \times 0.6 = 1.2$	1
$2 \times 0.2 = 0.4$	0
$2 \times 0.4 = 0.8$	0
$2 \times 0.8 = 1.6$	1
$2 \times 0.6 = 1.2$	1
$2 \times 0.2 = 0.4$	0

Процедура преобразования в примере 2 носит характер бесконечного повторения группы одинаковых операций и результатов. Поэтому ограничимся восемью разрядами. Тогда получим $0.3_{10} = 0.0100\ 1100_2$.

Задания для самопроверки

13. Какую из указанных операций необходимо выполнять при преобразовании целой части числа из одной системы счисления в другую: а) сложение, б) вычитание, в) умножение или г) деление?

14. Какую операцию необходимо выполнять при преобразовании дробной части числа из одной системы счисления в другую: а) сложение, б) вычитание, в) умножение или г) деление?

15. Преобразуйте в двоичные эквиваленты следующие десятичные числа: а) 23, б) 105, в) 32, г) 15, д) 206, е) 128, ж) 63, з) 29, и) 12.125, к) 16.375, л) 5.015625, м) 2.5.

2.5. Восьмеричная система счисления

Восьмеричной системой счисления является система с основанием 8. В ней используются следующие символы: 0, 1, 2, 3, 4, 5, 6 и 7. В каких случаях применяется эта система? Известно, что в электронных системах не используются восьмиуровневые сигналы; что же касается цифровых схем, то они оперируют сигналами двух уровней. Восьмеричная система счисления не нужна вычислительным машинам, она удобна как компактная форма записи чисел и широко применяется пользователями больших, мини- и микро-ЭВМ.

Обратимся вновь к двоичным числам. При работе с такими числами мы сталкиваемся с серьезной проблемой – их длина слишком велика. Так, многие микропроцессоры оперируют 8-битовыми словами. Но 8-разрядное двоичное число позволяет выразить десятичные величины в пределах от 0 до 255, в то время как 8-разрядное десятичное число охватывает диапазон величин от 0 до 99 999 999. Двоичное представление величин этого диапазона оказалось бы очень длинным – состоящим из 27 бит! Такие длинные числа чрезвычайно трудно читать. Тогда и приходит на помощь восьмеричная система счисления, позволяющая выполнять компактную запись длинных двоичных чисел.

Поскольку $2^3 = 8$, то каждый восьмеричный символ (от 0 до 7) может быть представлен 3-битовым числом. Причем таких чисел (от 000 до 111) восемь, как и символов восьмеричной системы. В табл. 2.2 приводятся некоторые восьмеричные числа и их двоичные и десятичные эквиваленты.

Таблица 2.2. **Восьмеричные числа и их двоичные и десятичные эквиваленты**

Восьмеричное число	Двоичное число	Десятичное число
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
10	1000	8
11	1001	9
12	1010	10
13	1011	11
14	1100	12
15	1101	13
16	1110	14
17	1111	15
20	1 0000	16
62	11 0010	50
100	100 0000	64
156	110 1110	110
200	1000 0000	128

Для представления двоичных чисел восьмеричными цифрами биты объединяют в группы по три, начиная с младшего значащего бита. Каждую группу битов преобразуют в восьмеричный эквивалент. Например, двоичное число 1010101111101_2 записывается восьмеричными цифрами как число 25375_8. Подобным же образом двоичное число 11000111_2 представляется эквивалентным восьмеричным числом 307_8, которое читать значительно легче. В обоих примерах прежде чем разбить биты на группы по три, двоичные числа были дополнены слева незначащими нуля-

ми. Такое дополнение не изменяет значения числа.

Дробную часть двоичного числа тоже можно представить восьмеричным эквивалентом. И в этом случае биты следует объединять в группы по три, однако формирование групп необходимо начинать со старшего значащего бита, ближайшего к двоичной точке. Незначащие нули при необходимости добавляются справа. Например, двоичная дробь 0.0101101_2 может быть представлена ее восьмеричным эквивалентом 0.264_8. При выполнении подобного преобразования для двоичного числа 1011.0101_2 после добавления слева и справа по два незначащих нуля (001011.010100_2) получим восьмеричное число 13.24_8. Однако следует быть внимательным и помнить, что $13.24_8 \neq 13.24_{10}$. Во избежание подобных ошибок используйте различные способы чтения восьмеричных и десятичных чисел. Так, восьмеричное число 13.24 рекомендуется произносить «один – три – точка – два – четыре», – в отличие от привычного чтения десятичного числа 13.24, а именно «тринадцать целых двадцать четыре сотых».

Как следует из рассмотренных примеров, восьмеричное представление двоичного числа позволяет существенно сократить длину записи числа. В случае 8-разрядных двоичных чисел, широко используемых в микропроцессорах, диапазон их значений простирается от $0000\ 0000_2$ до $1111\ 1111_2$ или, что то же самое, от 000_8 до 377_8. Поскольку такие двоичные числа содержат только 8 бит, их восьмеричные эквиваленты никогда не достигают значения 777_8. Для представления последнего требуется 9 бит.

Задания для самопроверки

16. Какие символы используются в восьмеричной системе счисления: а) 0 и 1, б) $1 \div 8$, в) $0 \div 9$ или г) $0 \div 7$?

17. До какого количества символов из указанного здесь сокращается двоичная запись байта при использовании восьмеричных символов: до а) 4, б) 3, в) 2 или г) 1?

18. Представьте восьмеричными эквивалентами следующие двоичные числа:

а) 101 б) 1111 1111 в) 1110 1101 г) 0000 0000 д) 1000 0000 е) 0101 0101 ж) 0000.0010 з) 111.111 и) 0110.0110 к) 1000.0001

19. Являются ли числа 137783 и 13777 восьмеричными? И если являются, то почему?

2.6. Шестнадцатеричная система счисления

После знакомства с восьмеричной системой счисления *шестнадцатеричная система* представляется естественной и несложной. Здесь используются следующие 16 символов: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E и F. Если выбор восьмеричной системы счисления обусловлен тем, что число 8 есть число 2 в третьей степени, то практическое использование шестнадцатеричной системы объясняется аналогичными причинами: число 16 есть число 2 в четвертой степени. Иначе говоря, шестнадцатеричную цифру можно использовать как средство сокращенной записи 4-разрядного двоичного числа. В табл. 2.3 приводятся примеры шестнадцатеричных чисел и их двоичных и десятичных эквивалентов.

Процедура преобразования двоичного числа в шестнадцатеричное довольно проста. Биты, начиная с младшего значащего бита (расположенного рядом с двоичной точкой), объединяются в группы по четыре. Каждой группе подбирается соответствующий шестнадцатеричный символ.

Рассмотрим два примера. Чтобы преобразовать двоичное число 10101011111101_2, необходимо добавить слева два незначащих нуля с целью формирования битов в группы по четыре: 0010 1010 1111 1101. Заменив каждую группу битов соответствующим шестнадцатеричным символом, получим число $2AFD_{16}$. В результате преобразования двоичного числа 11000111_2 в шестнадцатеричное получим число $C7_{16}$. Сравнив исходные данные и результаты обоих примеров, нетрудно заметить, что шестнадцатеричная форма записи числа много проще и легче воспринимается, чем двоичная. К аналогичному выводу приводит сопоставление восьмеричной записи числа с двоичной. Действительно, для рассматриваемых примеров

Таблица 2.3. **Шестнадцатеричные числа и их двоичные и десятичные эквиваленты**

Шестнадца-теричное число	Двоичное число	Десятичное число
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15
10	1 0000	16
11	1 0001	17
12	1 0010	18
13	1 0011	19
14	1 0100	20
15	1 0101	21
16	1 0110	22
17	1 0111	23
18	1 1000	24
19	1 1001	25
1A	1 1010	26
1B	1 1011	27
1C	1 1100	28
1D	1 1101	29
1E	1 1110	30
1F	1 1111	31
20	10 0000	32
32	11 0010	50
40	100 0000	64
6E	110 1110	110
80	1000 0000	128

имеем $101010111111101_2 = 25375_8 = 2AFD_{16}$ и $11000111_2 = 307_8 = C7_{16}$.

Следует помнить, что шестнадцатеричные и восьмеричные числа – это только способ представления двоичных чисел, которыми фактически оперирует микропроцессор. При этом шестнадцатеричная запись числа предпочтительнее восьмеричной, поскольку микропроцессоры манипулируют словами длиной 4, 8, 16 или 32 бит, т. е. словами, длина которых кратна 4. Ведь для записи двоичных чисел такой длины в восьмеричной системе счисления приходится добавлять незначащие нули, чтобы длина стала кратной трем.

В подобных добавлениях нет необходимости при использовании шестнадцатеричного представления слов микропроцессора. Например, 8-битовое число можно разделить на два 4-битовых, каждое из которых представляется одним шестнадцатеричным символом.

Простота соотношений между шестнадцатеричной и двоичной формами записи чисел – причина значительно большей распространенности шестнадцатеричной системы счисления по сравнению с восьмеричной. Последняя нашла широкое применение в семействе ЭВМ PDP-8 фирмы DEC (Digital Equipment Corporation), поскольку эти машины оперируют 12-битовыми словами, которые легко представляются в восьмеричной системе счисления.

Преобразование двоичных дробей в шестнадцатеричные осуществляется по правилам, аналогичным для преобразования этих дробей в восьмеричные. Для этой цели биты дробной части, начиная со старшего значащего бита (расположенного справа от двоичной точки), группируются по четыре. Добавление незначащих нулей осуществляется по мере необходимости. Например, для преобразования двоичной дроби 0.0101101_2 в шестнадцатеричную биты группируют по четыре: 0.0101 1010. Затем каждую группу заменяют шестнадцатеричным символом, получая в результате $0.5A_{16}$. Подобным образом, преобразуя число 1101.0111_2, составляют группы 1101. 0111, и после их замены шестнадцатеричными символами формируется число $D.7_{16}$.

Задания для самопроверки

20. Какие символы шестнадцатеричной системы счисления используются для пред-

ставления двоичных чисел: а) $0 \div 10$, б) $0 \div F$, в) $0 \div 7$ или г) 0 и 1?

21. В какой системе счисления представлено число 01C34: десятичной, восьмеричной или шестнадцатеричной? Дайте объяснение.

22. Во сколько раз уменьшается длина записи числа при переходе от двоичной формы представления к шестнадцатеричной [в а) 4, б) 3, в) 2 или г) 1]?

23. Преобразуйте в шестнадцатеричные эквиваленты следующие двоичные числа: а) 101 б) 1111 1111 в) 1110 1101 г) 0000 0000 д) 1000 0000 е) 0101 0101 ж) 0000 0010 з) 111.111 и) 0110.0110 к) 1000.0001

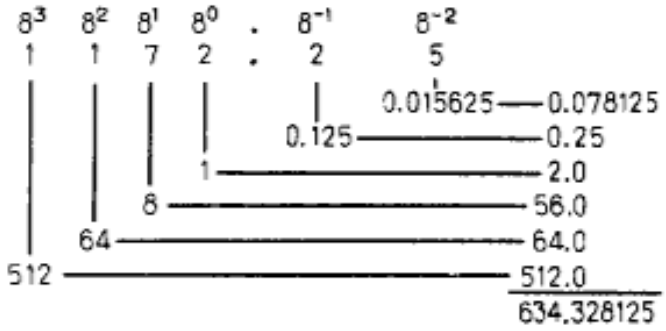
24. Сравните и проанализируйте результаты выполнения заданий пп. 23 и 18.

2.7. Восьмеричная, десятичная и шестнадцатеричная системы счисления. Преобразования из одной системы в другую

При работе с микропроцессорами возникает необходимость преобразования десятичных чисел в двоичные и обратного преобразования. Часто оказывается желательным представление двоичных чисел в восьмеричной или шестнадцатеричной форме. Как следствие этого, возникает и обратная задача: преобразование восьмеричных или шестнадцатеричных чисел в двоичные. Может также потребоваться преобразовать восьмеричные или шестнадцатеричные числа в десятичные.

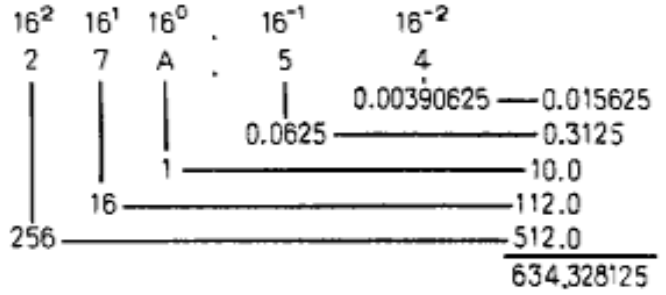
Преобразование восьмеричных или шестнадцатеричных чисел в десятичные схоже с преобразованием этих чисел в их двоичные эквиваленты. Каждой позиции числа присваивается определенный вес. Затем значение веса позиции умножается на цифру, занимающую эту позицию. Результаты операций умножения, выполненных для всех позиций числа, суммируются. Следующие ниже примеры демонстрируют преобразование чисел из восьмеричной и шестнадцатеричной систем в десятичную.

Пример. Представить в десятичной системе восьмеричное число 1172.25_8:



Результат: $1172.25_8 = 634.328125_{10}$.

Пример. Представить в десятичной системе шестнадцатеричное число $27A.54_{16}$:



Результат: $27A.54_{16} = 634.328125_{10}$.

Описанная процедура в действительности весьма проста. Что же касается дробной части, то на примере двоичных дробей можно было убедиться, что число может оказаться весьма длинным. Поэтому часто приходится прибегать к округлению.

Рассмотрим теперь преобразование десятичных чисел в восьмеричные или шестнадцатеричные. Можно воспользоваться процедурой, подобной той, с помощью которой осуществлялось преобразование десятичных чисел в двоичные, поскольку она в определенном смысле универсальна. Для преобразования десятичного числа в восьмеричное или шестнадцатеричное вместо деления (умножения для дробей) исходного числа на 2 необходимо выполнить деление (умножение) на 8 или 16. Остатки (целые части произведений) используются для формирования результата.

Ниже приводятся два примера преобразования десятичного числа в восьмеричное и шестнадцатеричное. Процедуры обработки целой и дробной частей десятичного числа отличаются друг от друга и выполняются порознь. Результаты объединяются

при формировании восьмеричного или шестнадцатеричного эквивалентов.

Пример 1. Преобразовать десятичное число 634.328125_{10} в восьмеричное:

Преобразование целой части				Преобразование дробной части	
Шаг	Деление	Частное	Остаток	Умножение	Произведение (целая часть)
1	634/8	79	2 (МЗР)	$8 \times 0.328125 = 2.625$	2 (СЗР)
2	79/8	9	7	$8 \times 0.625 = 5.0$	5 (МЗР)
3	9/8	1	1		
4	1/8	0	1 (СЗР)		

Результат: $634_{10} = 1172_8$ $0.328125_{10} = 0.25_8$
Общий результат: $634.328125_{10} = 1172.25_8$

Пример 2. Преобразовать десятичное число 634.328125_{10} в шестнадцатеричное:

Преобразование целой части				Преобразование дробной части	
Шаг	Деление	Частное	Остаток	Умножение	Произведение (целая часть)
1	634/16	39	$10_{10} = A_{16}$ (МЗР)	$16 \times 0.328125 = 5.25$	5 (СЗР)
2	39/16	2	7	$16 \times 0.25 = 4.0$	4 (МЗР)
3	2/16	0	2 (СЗР)		

Результат: $634_{10} = 27A_{16}$ $0.328125_{10} = 0.54_{16}$
Общий результат: $634.328125_{10} = 27A.54_{16}$

Отметим, что преобразование десятичных чисел в их восьмеричные или шестнадцатеричные эквиваленты выполняется с целью получения более компактной формы представления исходных десятичных данных. Преобразуя числа 1172.25_8 и $27A.54_{16}$ в их двоичные эквиваленты, можно убедиться, что это одно и то же число:

$$\underset{001}{1}\ \underset{001}{1}\ \underset{111}{7}\ .\ \underset{010}{2}\ \underset{010}{2}\ \underset{101}{5_8}$$
$$\underset{0010}{2}\ \underset{0111}{7}\ \underset{1010}{A}\ .\ \underset{0101}{5}\ \underset{0100}{4_{16}}$$

Безусловно, существуют также процедуры преобразования восьмеричных чисел в шестнадцатеричные, и наоборот. Однако необходимость в их использовании возникает крайне редко. Но если такая проблема появилась, одним из способов ее решения является представление исходного числа в двоичной форме, а затем преобразование полученного двоичного числа в требуемую систему счисления.

Задания для самопроверки

25. Какому преобразованию из указанных ниже подобно преобразование десятичных чисел в восьмеричные или шестнадцатеричные: а) двоичных чисел в восьмеричные, б) шестнадцатеричных чисел в двоичные, в) десятичных чисел в двоичные или г) двоичных чисел в шестнадцатеричные?

26. Приводимые ниже десятичные числа преобразуйте в двоичные, восьмеричные и шестнадцатеричные:

а) 126 б) 4 в) 16 г) 63 д) 101 е) 12 ж) 1 з) 127 и) 9.25 к) 64.015625

27. Как представить вещественное число, содержащее целую и дробную части, в системе счисления с основанием 4? Какие символы следует использовать в этом случае?

Упражнения

2.1. Что называется основанием системы счисления: наибольшее возможное для использования число, количество употребляемых в системе символов, число 8 в десятичной системе или число F в шестнадцатеричной?

2.2. Какова роль символа 4 в числе 426: старшая значащая цифра, младший значащий бит, старший значащий бит или младшая значащая цифра?

2.3. Чему равен вес разряда тысячных долей десятичного числа: 10^0, 10^{-1}, 10^{-2} или 10^{-3}?

2.4. Какие символы: 0 ÷ 9, 0 и 1, 0 ÷ F или 0 до 7 – используются при записи чисел в двоичной системе счисления?

2.5. Чему равен результат возведения числа 2 в степени 0, 1, 2, ..., 11 и 12?

2.6. Какое количество различных десятичных чисел можно представить 8-разрядным двоичным числом: 8, 99 999 999, 256 или 1024?

2.7. Что такое бит: двоичное приращение, двоичная цифра, младшее или старшее значащее число?

2.8. Преобразуйте в десятичные числа следующие двоичные числа:

а) 1111 б) 1111 1111 в) 0100 г) 0000 0000 д) 1101 е) 1101 1101 ж) 0001 з) 0000 0001 и) 0111 к) 0011 1111 л) 0.0001 м) 101.11111 н) 1110.1110 о) 00.01011

2.9. Преобразуйте в двоичные числа следующие десятичные числа: 128, 12, 75, 256, 31, 30, 56, 29, 255, 1024, 10.5, 16.015625, 12.03, 17.7.

2.10. Преобразуйте двоичные числа, заданные в п. 2.8, в эквивалентные восьмеричные, десятичные и шестнадцатеричные числа.

2.11. Преобразуйте десятичные числа, заданные в п. 2.9, в эквивалентные двоичные, восьмеричные и шестнадцатеричные числа.

Ответы на вопросы заданий для самопроверки

1. б. 2. в. 3. 27 329. 4. г. 5. б. 6. Нет, поскольку символ 2 не является двоичным символом. 7. Младший значащий бит (МЗБ). 8. Двоичная цифра. 9. в. 10. 1111_2, поскольку это число равно 15_{10}. 11. б. 12. а) 182; б) 80; в) 30; г) 128; д) 255; е) 129; ж) 126; з) 63; и) 0.3125; к) 4.375. 13. г. 14. в. 15. а) 10111; б) 1101001; в) 100000; г) 1111; д) 11001110; е) 10000000; ж) 111111; з) 11101; и) 1100.001; к) 10000.011; л) 101.000001; м) 10.1. 16. г. 17. б. 18. а) 5; б) 377; в) 355; г) 000; д) 200; е) 125; ж) 00.1; з) 7.7; и) 06.30; к) 10.04.

19. Первое число не является восьмеричным, так как 8 не служит символом в восьмеричной системе. Второе число может быть восьмеричным, поскольку символы, использованные при его записи, принадлежат этой системе. Однако по той же причине оно может относиться и к десятичной системе счисления. 20. б.

21. В шестнадцатеричной, поскольку в других системах счисления символ С не используется.

22. а. 23. а) 5; б) FF; в) ED; г) 00; д) 80; е) 55; ж) 0.0; з) 7.E; и) 6.6; к) 8.1.

24. Записи числа 5 (п. «а») в восьмеричной и шестнадцатеричной системах совпадают. В любой системе 0 обозначает 0 (п. «г»). В остальных случаях шестнадцатеричная форма записи более компактна по сравнению с восьмеричной. 25. в.

26.

	Двоичные числа	Восьмеричные числа	Шестнадцатеричные числа
а)	01111110	176	7E
б)	100	4	4
в)	10000	20	10
г)	111111	77	3F
д)	1100101	145	65
е)	1100	14	C
ж)	1	1	1
з)	01111111	177	7
и)	1001.01	11.2	9.4
к)	1000000.000001	100.01	40.04

27. Целую часть десятичного числа нужно разделить на 4. Остаток использовать в качестве младшего значащего разряда. Повторять деление до тех пор, пока результат не станет равным 0. Последний остаток – старший значащий разряд искомого результата. Затем дробную часть исходного числа следует умножить на 4. Целую часть произведения использовать в качестве старшего значащего разряда дробной части искомого результата. Дробную часть произведения умножить на 4 и т. д. Умножение повторять до тех пор, пока не будет достигнута требуемая точность.

В системе счисления с основанием 4 используются следующие символы: 0, 1, 2 и 3.

Глава 3. *Внутреннее построение микропроцессора*

В этой главе читатель знакомится с внутренним устройством микропроцессора. Сначала описываются его структурная схема и ее три основных узла: арифметическо-логическое устройство (АЛУ), регистры данных и устройство управления. Показывается, что АЛУ выполняет арифметические и логические операции над данными, изменяя их; регистры данных являются быстродействующей внутренней памятью, каждый из них способен хранить одно слово данных. Почти каждый микропроцессор имеет шесть таких регистров; назначение дополнительных регистров, которыми может располагать микропроцессор, описывается в гл. 7. Затем в данной главе рассматривается устройство управления. В заключение анализируются функции внутренней шины микропроцессора, т. е. операции по перемещению данных из одного узла в другой. Далее, в гл. 6 демонстрируется, что принципы функционирования этой шины реализованы и в микропроцессорной системе в целом.

3.1. Структурная схема микропроцессора

Структурная схема микропроцессора дает возможность наглядно рассмотреть его работу по выполнению двух основных функций: обработке и манипулированию данными. Использование такой схемы часто существенно облегчает понимание того, как микропроцессор решает поставленные задачи. В дальнейшем мы будем пользоваться структурной схемой, представленной на рис. 3.1. Изображенный на этом рисунке 8-разрядный микропроцессор с регистрами приведен в качестве типичного примера микропроцессора и не является какой-либо конкретной моделью, выпускаемой промышленностью. На практике при работе с тем или иным микропроцессором пользователю должна быть предоставлена соответствующая документация, включающая и структурную схему. В каждом случае наличие такой схемы облегчает понимание архитектуры микропроцессорной вычислительной системы.

Техническая документация, прилагаемая к микропроцессору, содержит также и так называемые средства программирования, которые на первый взгляд похожи на структурную схему. Однако в действительности они существенно различаются. В данной главе речь идет о структурной схеме. Отличительные особенности средств, используемых для программирования, по сравнению со структурной схемой описываются в последующих главах.

Согласно рис. 3.1, микропроцессор состоит из трех основных блоков: АЛУ, нескольких регистров и устройства управления. Для передачи данных между этими блоками микропроцессора используется внутренняя шина данных. АЛУ рассматривается в разд. 3.2, а затем описываются регистры, устройство управления и внутренняя шина данных.

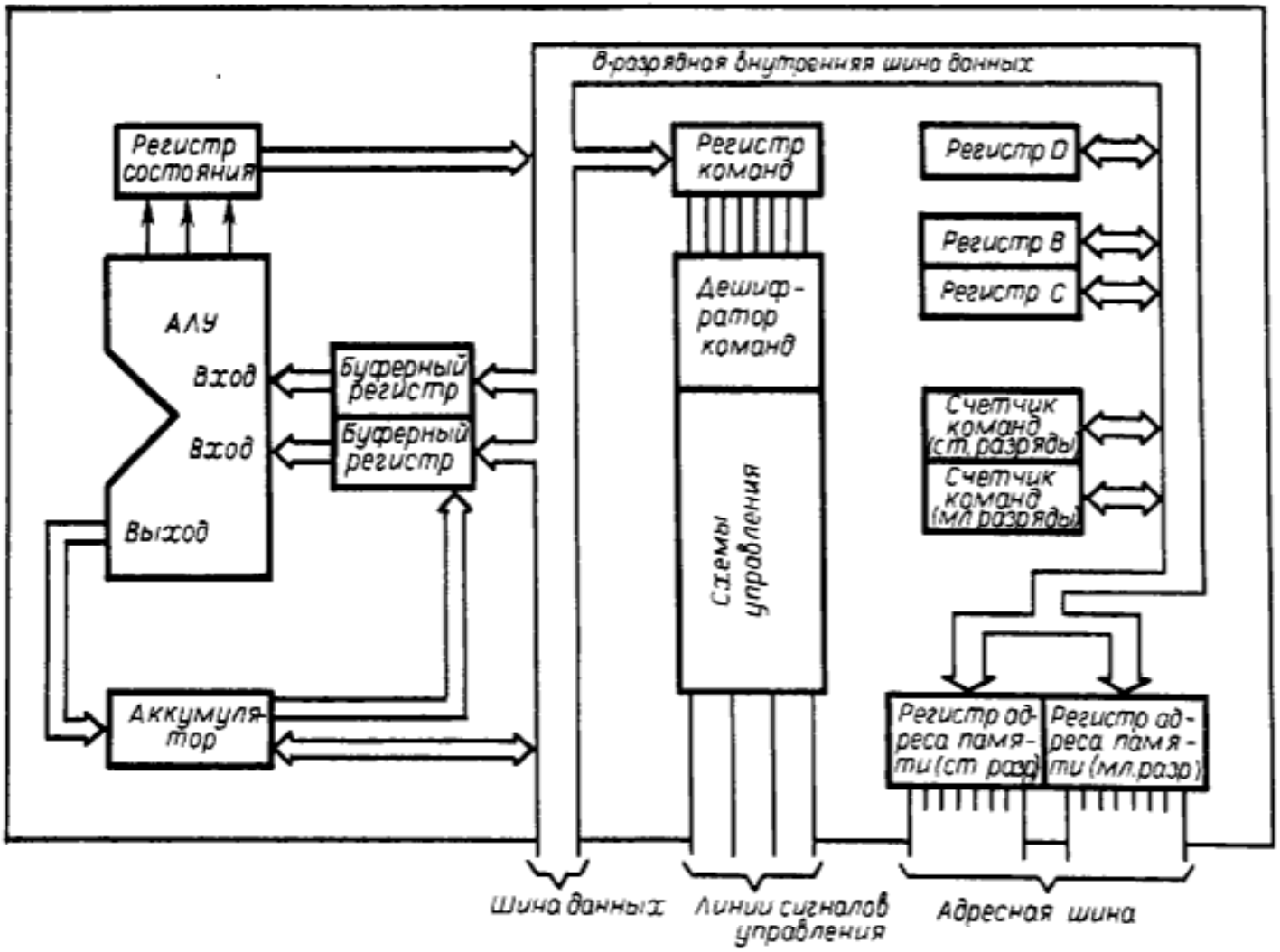


Рис. 3.1. Структурная схема 8-разрядного микропроцессора.

3.2. АЛУ

АЛУ выполняет одну из главных функций микропроцессора – обработку данных. В соответствии с рис. 3.1 АЛУ имеет два *входных порта*, обозначенных как «Вход», и один *выходной порт* – «Выход». Назначение входного порта – ввод слова данных в АЛУ, а выходного порта – вывод такого слова. Как правило, подобные логические схемы имеют один или несколько входных портов и единственный выходной порт. Оба входных порта снабжены *буферами*, роль которых выполняют регистры временного хранения данных (*буферные регистры*). Каждый порт соединен со своим буферным регистром, способным хранить для АЛУ одно слово данных. Более подробное описание этих регистров дано в разд. 3.9.

Два входных порта позволяют АЛУ принимать данные или с внутренней шины данных микропроцессора, или из специального регистра, именуемого *аккумулятором*. Единственный выходной порт АЛУ предоставляет последнему возможность пересылать слово данных в аккумулятор.

Аккумулятор предназначен для хранения слова данных, посланного в него из выходного порта АЛУ или извлеченного из памяти. Когда, например, АЛУ складывает два слова данных, одно из них находится в аккумуляторе. После выполнения сложения результат – слово данных – посылается в аккумулятор на хранение.

АЛУ оперирует одним или двумя словами в зависимости от вида выполняемой операции; соответственно он использует и входные порты. Так, например, поскольку для сложения требуются два слова данных, то эту операцию АЛУ производит, пользуясь обоими входными портами. А при инвертировании слова данных АЛУ ограничивается одним входным портом. Эта операция сводится к замене в слове данных всех двоичных нулей на двоичные единицы, а всех

двоичных единиц на двоичные нули. Поскольку операция выполняется над одним словом, достаточно подсоединить к аккумулятору один входной порт.

АЛУ необходимо использовать в тех случаях, когда требуется изменить или проверить значение слова данных. Перечень функций АЛУ зависит от типа микропроцессора и различен для машин разных типов. Некоторые АЛУ способны выполнять много различных операций, у других набор операций ограничен небольшим числом. Функции АЛУ определяют архитектуру микропроцессора в целом. Типичными операциями, выполняемыми АЛУ большинства микропроцессоров, являются следующие: СЛОЖЕНИЕ, ВЫЧИТАНИЕ, И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ, ИНВЕРСИЯ, СДВИГ ВПРАВО, СДВИГ ВЛЕВО, ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ, ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ.

Задания для самопроверки

1. Каково назначение структурной схемы микропроцессора: а) описать детальную конфигурацию схем на вентилях и триггерах с целью конструирования микропроцессора, б) описать соединения логической схемы микропроцессора с памятью и устройствами ввода-вывода, в) показать логические устройства, с помощью которых можно решать поставленную задачу, г) реализация всех перечисленных функций?

2. Какую из перечисленных далее функций АЛУ не выполняет: а) сложение, б) сдвиг влево, в) включение питания или г) инверсию?

3. АЛУ имеет два входа. К какому узлу микропроцессора, кроме аккумулятора, они подключены согласно схеме, изображенной на рис. 3.1: а) счетчику команд, б) внутренней шине данных, в) устройству управления или г) регистру адреса памяти?

4. В чем заключается основная функция АЛУ: а) выполнять операции сложения, б) служить источником сигналов для аккумулятора, в) изменять данные посредством арифметических или логических операций или г) выполнять все перечисленные функции?

3.3. Регистры микропроцессора

Регистры являются важной составной частью любого микропроцессора. Они участвуют в реализации основных логических функций микропроцессора независимо от количества регистров. В данном разделе ограничимся рассмотрением шести основных регистров. Другие регистры будут описаны ниже.

Каждый регистр микропроцессора может использоваться для временного хранения одного слова данных. Некоторые регистры имеют специальное назначение, другие – многоцелевое. Регистры последнего типа называются *регистрами общего назначения* и могут использоваться программистом по его усмотрению. Благодаря таким возможностям лица, более искушенные в программировании, добиваются лучших результатов.

Количество и назначение регистров в микропроцессоре зависит от архитектуры последнего. Однако почти все микропроцессоры имеют шесть основных регистров: *состояния, буферные, команд, адреса памяти, счетчик команд* и *аккумулятор*. Остальные регистры предназначены для упрощения и облегчения работы программиста.

В процессе ознакомления с каждым из основных регистров следует обращать внимание на то, какое влияние оказывает именно этот регистр на данные, проходящие «сквозь» микропроцессор. Возможно, у вас вызовет удивление то, что некоторые основные регистры иногда не используются в качестве средства программирования. Это объясняется не их физическим отсутствием, а тем, что программисту не предоставляются средства изменения содержимого этих регистров. Еще раз подчеркнем, что каждый основной регистр микропроцессора имеет свое определенное назначение, а поэтому имеется в его составе. Только понимание влияния каждого основного регистра на поток данных в микропроцессоре позволяет получить правильное представление о функционировании последнего.

3.4. Аккумулятор

Аккумулятор – главный регистр микропроцессора при различных манипуляциях с данными. Большинство арифметических и логических операций осуществляется путем использования АЛУ и аккумулятора. Любая из таких операций над двумя словами данных (*операндами*) предполагает размещение одного из них в аккумуляторе, а другого в памяти или еще каком-либо регистре. Так, при сложении двух слов, называемых условно A и B и расположенных в аккумуляторе и памяти соответственно, результирующая сумма C загружается в аккумулятор, замещая слово A. Результат операции АЛУ тоже обычно размещается в аккумуляторе. Следует помнить, что исходное содержимое последнего при этом теряется.

Операцией другого типа, использующей аккумулятор, является программируемая передача данных из одной части микропроцессора в другую. Речь идет о пересылке данных между портом ввода-вывода и областью памяти, между двумя областями памяти и т.п. Выполнение операции «программируемая передача данных» осуществляется в два этапа: сначала выполняется пересылка данных из источника в аккумулятор, а затем из аккумулятора – в пункт назначения.

Выше было показано, что микропроцессор позволяет использовать АЛУ для объединения данных в аккумуляторе с другими данными. Однако микропроцессор может выполнять некоторые действия над данными непосредственно в аккумуляторе. Например, аккумулятор может быть очищен путем записи двоичных нулей во все его разряды, установлен в единичное состояние посредством записи двоичных единиц во все разряды. Содержимое аккумулятора можно сдвигать влево или вправо, получать его инвертированное значение, а также выполнять другие операции. Ряд команд, для реализации которых используется аккумулятор, описывается при рассмотрении системы команд микропроцессора.

Аккумулятор является наиболее универсальным регистром микропроцессора: для выполнения любой операции над данными прежде всего необходимо поместить их в аккумулятор. Как показано на рис. 3.1, данные поступают в него с внутренней шины данных микропроцессора. В свою очередь аккумулятор может посылать данные на эту шину. Следует обратить внимание на наличие буфера на пути прохождения данных из аккумулятора в АЛУ. (Назначение соответствующего буферного регистра описывается в разд. 3.9.)

Количество разрядов аккумулятора соответствует длине слова микропроцессора, т.е. 8 бит. Однако некоторые микропроцессоры имеют *аккумуляторы двойной длины*. Такой аккумулятор можно рассматривать или как одно устройство, или как два отдельных аккумулятора. В первом случае второй аккумулятор пары используется для записи дополнительных битов, появляющихся при выполнении некоторых арифметических операций. Например, при умножении двух 8-битовых слов результат – 16-битовое слово – размещается в аккумуляторе двойной длины.

У некоторых микропроцессоров имеется группа аккумуляторов. Если их, например, два – аккумуляторы A и B, то микропроцессор должен располагать двумя различными командами для загрузки в них данных с выхода АЛУ: одной командой – для записи данных в аккумулятор A, другой – для записи данных в аккумулятор B. Кроме того, должны быть две соответствующие команды очистки этих аккумуляторов.

Преимущество «многоаккумуляторных» микропроцессоров по сравнению с «одноаккумуляторными» в том, что первые предоставляют возможность выполнения операций с передачей данных от аккумулятора к аккумулятору. Данные могут временно храниться в одном аккумуляторе, пока другой используется для выполнения каких-либо иных действий. Когда вновь возникает необходимость в данных, содержащихся в первом аккумуляторе, пересылать их не нужно, поскольку они уже находятся там.

Что же касается функционирования микропроцессора с одним аккумулятором, то операции выполняются именно над его содержимым, и по завершении очередной операции результат приходится записывать в память или в другой регистр. Во

многих случаях это совпадает с намерениями программиста. Однако в некоторых случаях наличие двух аккумуляторов может сократить число необходимых операций.

Задания для самопроверки

5. Для большинства арифметических и логических операций, выполняемых микропроцессором, требуются два «участника» операции – два операнда. Один из них находится в памяти или регистре. Укажите, где местоположение другого: а) в аккумуляторе, б) в счетчике команд, в) в регистре адреса памяти или г) в регистре команд?

6. Аккумулятор соединен с другими блоками микропроцессора внутренней шиной данных. Чем располагает аккумулятор для этих целей: а) 8 разрядами, б) 16 разрядами, в) входными и выходными портами или г) линиями с поданными на них двоичными нулями?

7. Что означает понятие «сбросить» или «очистить» аккумулятор: а) выполнить установку всех разрядов аккумулятора в 0, б) закоротить его выходы, в) выполнить установку всех разрядов аккумулятора в 1 или г) прекратить использование аккумулятора?

8. Предположим, что в вашем распоряжении имеется новый 16-разрядный микропроцессор с аккумулятором двойной длины. Вы предполагаете, что в этом аккумуляторе можно хранить результаты операций АЛУ. Какова длина данных, которые можно загружать в аккумулятор: а) 1 байт, б) 2 байт, в) 3 байт или г) 4 байт?

9. Что используется для временного хранения данных, подлежащих программируемой передаче между входным портом 200_8 и областью памяти по адресу 7234_8: а) счетчик команд, б) АЛУ, в) область памяти по адресу 7233_8 или г) аккумулятор?

3.5 Счетчик команд

Счетчик команд – это один из наиболее важных регистров микропроцессора. Как известно, программа – это последовательность команд, хранимых в памяти микро-ЭВМ и предназначенных для того, чтобы инструктировать машину, как решать поставленную задачу. Для корректного выполнения последней команды должны поступать в строго определенном порядке. На счетчике команд лежит ответственность следить за тем, какая команда выполняется, а какая подлежит выполнению следующей. Часто счетчик команд имеет намного больше разрядов, чем длина слова данных микропроцессора. Так, в большинстве 8-разрядных микропроцессоров, адресующихся к памяти объемом 65К, число разрядов счетчика команд равно 16. И на это имеются достаточно веские основания. В любой из 65 536 областей памяти микро-ЭВМ общего назначения может находиться информация о том или ином шаге программы, т.е. в пределах диапазона значений адресов от 0 до 65 535 программа может начаться и закончиться в любом месте. Чтобы обратиться по любому из этих адресов, счетчик команд должен располагать 16 двоичными разрядами. Еще раз подчеркнем, что где бы команды программы ни располагались, они следуют друг за другом в определенном порядке.

Согласно рис. 3.1, счетчик команд соединен с внутренней шиной данных микропроцессора. Теоретически этот счетчик может получать данные об адресах программы из любого блока микропроцессора, подключенного к внутренней шине. Однако на практике данные обычно поступают в счетчик команд из памяти микро-ЭВМ.

Когда микропроцессор начинает работать, по команде начальной установки в счетчик команд загружаются данные из области памяти, заданной проектировщиком микропроцессора. Перед пуском программы необходимо поместить начальный адрес для программы в область памяти, указанную проектировщиком. Когда программа начинает выполняться, первым значением содержимого счетчика команд является этот, заранее определенный адрес.

В отличие от аккумулятора счетчик команд не может выполнять операции различного типа. Набор команд, его использующих, крайне ограничен по сравнению с подобным набором для аккумулятора.

Перед выполнением программы счетчик

команд необходимо загрузить числом — адресом области памяти, содержащей первую команду программы. Обратите внимание, что на рис. 3.1 регистр адреса памяти и адресная шина расположены ниже счетчика команд. Адрес области памяти, содержащей первую команду программы, посылается из счетчика команд в регистр адреса памяти, после чего содержимое обоих регистров становится одинаковым. (Назначение регистра адреса памяти более детально описывается в разд. 3.6.) Длина регистра адреса памяти равна 16 разрядам.

Адрес местоположения первой команды программы посылается по адресной шине к схемам управления памятью, в результате чего считывается содержимое области с указанным адресом. Этим содержимым, конечно, должна быть команда. Память пересылает эту команду в специальный регистр блока микропроцессора, называемый регистром команд (см. разд. 3.7).

Для правильного понимания излагаемого ниже отметим следующее. После извлечения команды из памяти микропроцессор автоматически дает приращение содержимому счетчика команд. Это приращение счетчик команд получает как раз в тот момент, когда микропроцессор начинает выполнять команду, только что извлеченную из памяти. Следовательно, начиная с этого момента, счетчик команд «указывает», какой будет следующая команда. Счетчик команд содержит адрес следующей выполняемой команды на протяжении всего времени выполнения текущей команды. Об этом важно помнить, потому что, программируя работу микро-ЭВМ, вы можете столкнуться с необходимостью использования текущего значения счетчика команд. При этом необходимо четко сознавать, что в каждый данный момент счетчик команд указывает не текущую выполняемую команду, а команду, следующую за ней.

Счетчик команд может быть загружен иным содержимым при выполнении особой группы команд. Может возникнуть необходимость выполнить часть программы, которая «выпадает» из последовательности команд *основной*, или *главной*, *программы*. Например, такую часть программы, которую следует многократно повторять в процессе выполнения всей программы. Вместо того чтобы писать эту часть программы каждый раз, когда в ней возникает необходимость, такую запись можно сделать лишь один раз и возвращаться к ее повторному выполнению, отступая от указанной последовательности. Часть программы, выполняемая путем отступления от строгой последовательности команд главной программы, называется *подпрограммой*. После того как в счетчик команд записан начальный адрес подпрограммы, счетчик получает приращения по мере выполнения команд этой подпрограммы. Так продолжается до тех пор, пока не встретится команда возврата в главную программу. (Более подробно подпрограммы описываются в гл. 5.)

Задания для самопроверки

10. Диапазон адресов 16-разрядного микропроцессора равен $2^{16} = 65\,536$. Чему должно равняться число разрядов счетчика команд этого микропроцессора: а) 4, б) 8, в) 16 или г) 32?

11. Регистром какого типа является счетчик команд: а) специального назначения, б) общего назначения, в) памяти или г) всех перечисленных типов?

12. Какую команду программы указывает счетчик команд после извлечения из памяти очередной команды: а) последнюю выполненную, б) подлежащую выполнению следующей, в) текущую выполняемую или г) принадлежащую подпрограмме?

13. Какие действия выполняет счетчик команд после того, как произошла загрузка начального адреса: а) осуществляет просмотр адресов различных областей памяти, которые, возможно, понадобятся программе, б) переходит к области памяти с меньшим адресом после выполнения каждой команды, в) наращивает свое содержимое с целью указания адреса следующей выполняемой команды, после того как из памяти извлечена текущая выполняемая команда или г) переходит к области памяти с адресом 65 536?

3.6. Регистр адреса памяти

При каждом обращении к памяти микро-ЭВМ *регистр адреса памяти* указывает адрес области памяти, которая подлежит использованию микропроцессором. Регистр адреса памяти содержит двоичное число – адрес области памяти. Выход этого регистра называется *адресной шиной* и используется для выбора области памяти или в некоторых случаях для выбора порта ввода-вывода.

В течение подцикла выборки команды из памяти (см. разд. 3.5) регистры адреса памяти и счетчика команд имеют одинаковое содержимое, т. е. регистр адреса памяти указывает местоположение команды, извлекаемой из памяти. После декодирования команды счетчик команд получает приращение. Что же касается регистра адреса памяти, то он приращения не получает.

В течение подцикла выполнения команды содержимое регистра адреса памяти зависит от выполняемой команды. Если в соответствии с командой микропроцессор должен произвести еще одно обращение к памяти, то регистр адреса памяти подлежит вторичному использованию в процессе обработки этой команды. Для некоторых команд адресация к памяти не требуется. Такова, например, команда очистки аккумулятора. При обработке таких команд регистр адреса памяти используется лишь один раз – в течение подцикла выборки команды из памяти.

В большинстве микропроцессоров регистры адреса памяти и счетчика команд имеют одинаковое число разрядов. Как и счетчик команд, регистр адреса памяти должен располагать количеством разрядов, достаточным для адресации любой области памяти микро-ЭВМ. У большинства 8-разрядных микропроцессоров количество разрядов регистра адреса памяти равно 16. Такой регистр можно разделить на два отдельных регистра, каждый из которых имеет независимое подключение к шине данных микропроцессора. Один из этих регистров называют *регистром старшего байта (СБ)*, другой – *регистром младшего байта (МБ)*.

Поскольку регистр адреса памяти подключен к внутренней шине данных микропроцессора, он может загружаться от различных источников. Большинство микропроцессоров располагают командами, позволяющими загружать этот регистр содержимым счетчика команд, регистра общего назначения или какой-либо области памяти. Некоторые команды предоставляют возможность изменять содержимое регистра адреса памяти путем выполнения вычислений: новое значение содержимого этого регистра получается путем сложения или вычитания содержимого счетчика команд с числом, указанным в самой команде. Адресация такого типа получила название *адресации с использованием смещения*.

Задания для самопроверки

14. Регистр адреса памяти является указателем: а) содержимого области памяти, б) адреса области памяти, в) регистра памяти или г) части микропроцессора?

15. Пусть 16-разрядный микропроцессор может адресоваться к 4 194 304 областям памяти. Сколько разрядов должен иметь соответствующий регистр адреса памяти: а) 8, б) 16, в) 22 или г) 32?

16. Регистр адреса памяти соединен с внутренней шиной данных микропроцессора. Какой из перечисляемых ниже объектов может явиться источником данных для загрузки этого регистра: а) счетчик команд, б) регистр общего назначения, в) память или г) все перечисленные объекты?

17. Какой узел микропроцессора образуют выходы регистра адреса памяти: а) аккумулятор, б) внутреннюю шину данных, в) адресную шину или г) вход дешифратора команд?

3.7. Регистр команд

Регистр команд предназначен исключительно для хранения текущей выполняемой команды, причем эта функция реализуется микропроцессором автоматически с началом цикла выборка-выполнение, называемого также *машинным циклом*.

Как отмечалось выше, машинный цикл состоит из двух подциклов – выборки и выполнения. За исключением загрузки команды, в период подцикла выборки программист не

может по-другому использовать регистр команд. Согласно схеме на рис. 3.1, этот регистр соединен с внутренней шиной данных, однако он только принимает данные – посылать данные на шину он не может.

Хотя функции регистра команд ограниченны, роль его в работе микропроцессора велика, поскольку выход этого регистра является частью дешифратора команд.

Напомним еще раз последовательность реализации цикла выборка-выполнение. Сначала команда извлекается из памяти, затем счетчик команд настраивается на указание следующей команды, подлежащей выполнению. При извлечении команды из соответствующей области памяти копия команды помещается на внутреннюю шину данных и пересылается в регистр команд. После этого начинается подцикл выполнения команды, в течение которого дешифратор команд «читает» содержимое регистра команд, сообщая микропроцессору, что делать для реализации операций команды. (Более подробно функции дешифратора описаны в разд. 3.11.)

Число разрядов регистра команд зависит от типа микропроцессора: иногда оно совпадает с числом разрядов слова данных, в других случаях равно лишь 3 или 4.

Задания для самопроверки

18. Какая команда содержится в регистре команд в процессе выполнения текущей команды: а) предшествующая выполняемой, б) текущая выполняемая, в) подлежащая выполнению следующей или г) все перечисленные команды?

19. Чем определяется число разрядов регистра команд: а) архитектурой микропроцессора, б) разрядностью (8 или 16 разрядов) микропроцессора, в) размером адресуемой памяти или г) быстродействием микропроцессора?

20. Что является источником информации об адресе области памяти, содержимое которой загружается в регистр команд: а) аккумулятор, б) блок микропроцессора, в) предыдущая команда или г) счетчик команд?

3.8. Регистр состояния

Наличием *регистра состояния* подлинная вычислительная машина отличается от простого калькулятора. Указанный регистр предназначен для хранения результатов некоторых проверок, осуществляемых в процессе выполнения программы. Разряды регистра состояния принимают то или иное значение при выполнении операций, использующих АЛУ и некоторые регистры.

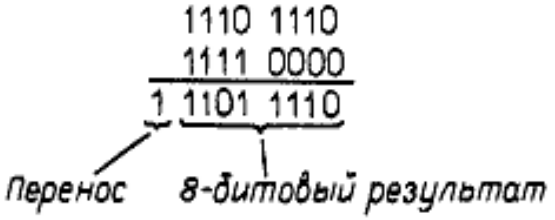
Запоминание результатов упомянутых проверок позволяет использовать программы, содержащие *переходы* (нарушения естественной последовательности выполнения команд).

При наличии в программе перехода выполнение команд начинается с некоторой новой области памяти, т. е. счетчик команд загружается новым числом. В случае условного перехода такое действие имеет место, если результаты определенных проверок совпадают с ожидаемыми значениями. Указанные результаты находятся в регистре состояния.

Возможности программирования с передачей управления (переходами) – отличительная характеристика вычислительной машины по сравнению с калькулятором. Регистр состояния предоставляет программисту возможность организовать работу микропроцессора так, чтобы при определенных условиях менялся порядок выполнения команд. Можно сказать, что микропроцессор принимает решение о том или ином продолжении хода вычислений в зависимости от указанных условий. Калькулятор такие решения принимать не может.

Итак, использование содержимого разрядов состояния привело к появлению нового набора команд микропроцессора. Эти команды предназначены для изменения хода выполнения программы в соответствии со значением, принимаемым тем или иным разрядом состояния. Традиционный способ использования этих специальных команд предполагает загрузку счетчика команд новым содержимым, если значение определенного разряда состояния становится равным 1. Команды условного перехода детально описываются в гл. 6..

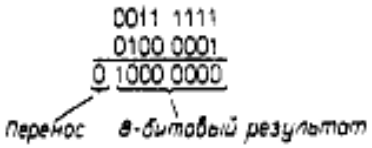
Как ранее отмечалось, при выполнении операций АЛУ разрядам регистра состояния присваиваются единичные значения. Типичным примером таких операций являются арифметические, при реализации которых возможно генерирование единичного бита переноса, формирование нулевого результата (двоичных нулей во всех разрядах) или и то и другое одновременно. Так, если при сложении двух 8-битовых чисел получается результат больше, чем 1111 1111, то появляется единичный бит переноса, который в свою очередь устанавливает в 1 одноименный разряд регистра состояния. Приведем пример такой операции:



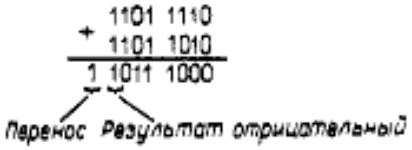
Появление бита переноса влечет за собой установку в 1 соответствующего разряда регистра состояния.

Рассмотрим еще два примера:

Пример 1



Пример 2



В первом примере переноса нет и значение соответствующего разряда регистра состояния равно 0, во втором примере имеет место и перенос (бит переноса равен 1), и признак отрицательного результата (бит отрицательного результата равен 1).

Если по окончании выполнения операции все разряды аккумулятора содержат биты, равные 0, то в регистре состояния бит нулевого результата становится равным 1. В рассматриваемом нами микропроцессоре этот бит может быть установлен в единичное состояние и некоторыми операциями, реализуемыми с участием регистров общего назначения. Например, часто требуется записать определенную величину в некоторый регистр (назовем его регистром D), а затем уменьшать ее на значение некоторой константы при каждом «проходе» через определенную точку программы. После каждого изменения содержимого этого регистра проверяется значение разряда нулевого результата в регистре состояния. Если содержимое регистра D оказывается равным 0, разряд нулевого результата устанавливается в 1. Программа (или ее часть), проверяющая наличие нуля в регистре D, продолжает повторяться до тех пор, пока в регистре состояния не будет обнаружено единичное значение разряда нулевого результата.

Проиллюстрируем сказанное примером использования регистра состояния для проверки содержимого некоторого регистра, получающего отрицательные приращения. Словесное описание соответствующей программы может иметь следующий вид:

1. Загрузить в регистр число 1100_2.
2. Уменьшить содержимое регистра на 1_2.
3. Проверить, равно ли единице значение разряда нулевого результата в регистре состояния.
4. Если нет, возвратиться к выполнению шага 2.
5. Если да, прекратить действия.

Кратко охарактеризуем некоторые наиболее общеиспользуемые разряды регистра состояния.

1. *Перенос/заем.* Данный разряд указывает, что последняя выполненная операция сопровождалась переносом или заемом (отрицательным переносом). Значение разряда переноса устанавливается равным 1, если в результате сложения двух двоичных чисел имеет место перенос из 8-го разряда результата. Отрицательный перенос (заем) фиксируется в регистре состояния при вычитании большего числа из меньшего.

2. *Нулевой результат.* Принимает единичное значение, если после окончания операции во всех разрядах регистра результата обнаружены двоичные нули. Установка этого разряда в 1 происходит не только при отрицательном приращении содержимого регистра, но и при любой другой операции, результат которой – число из двоичных нулей.

3. *Знаковый.* Принимает единичное значе-

ние, когда старший значащий бит содержимого регистра, предназначенного для записи результата операции, становится равным 1. При выполнении арифметических операций с числами в дополнительном коде единичное значение старшего значащего бита показывает, что в регистре находится отрицательное число.

Указанные три разряда состояния используются в большинстве микропроцессоров, а следовательно, и в гипотетическом микропроцессоре, описываемом в данной книге. Многие микропроцессоры располагают дополнительными разрядами состояния, применение которых не «стандартизовано». Для правильного использования таких разрядов программист должен хорошо понимать особенности работы с командами двоичной арифметики, поскольку разряды регистра состояния принимают единичные значения только в результате выполнения определенных арифметических операций. «Нестандартными» разрядами располагают не только операции, выполняемые с помощью АЛУ или с участием регистров. Подобные разряды регистра используются как индикаторы «включения» или «выключения» некоторых дополнительных программно-аппаратных средств микропроцессора. Поскольку такие разряды содержат информацию об аппаратных средствах, их значение следует анализировать перед принятием решения об использовании возможностей этих средств.

В некоторых микропроцессорах предусмотрены специальные команды для сброса, или очистки, всех разрядов состояния. Однако имеются микропроцессоры, допускающие «только чтение» содержимого регистра состояния. Чтобы понять, как пользоваться «нестандартными» разрядами регистра состояния, необходимо в каждом конкретном случае обращаться к технической документации интересующей вас модели микропроцессора.

Не все разряды регистра состояния используются микропроцессором. В неиспользуемые разряды регистра состояния обычно «навсегда» записываются двоичные единицы. Применительно к микропроцессору, рассматриваемому в данной книге, слово состояния – содержимое одноименного регистра – имеет только три используемых разряда (рис. 3.2). Пять младших разрядов

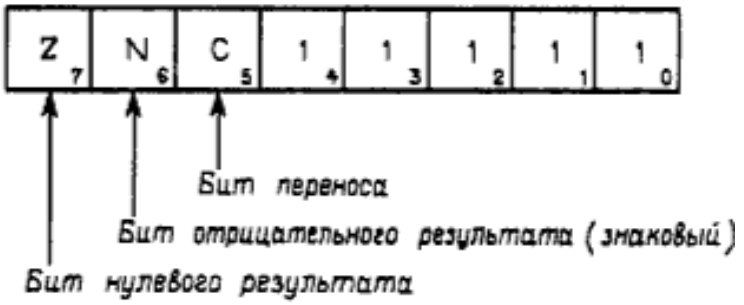


Рис. 3.2. Слово состояния микропроцессора с тремя модифицируемыми битами Z, N и C.

8-разрядного слова имеют постоянные единичные значения. В результате содержимое регистра состояния может быть загружено во внутреннюю шину данных микропроцессора, однако регистр состояния не имеет возможности принимать данные, поступающие по шине. Как следствие этого, если, например, записываемый в аккумулятор результат операции – положительное число без переноса, то слово состояния равно 0001 1111. Если же результат операции – отрицательное число без переноса, то в регистре состояния формируется число 0101 1111.

Задания для самопроверки

21. Сложите приводимые ниже 8-разрядные двоичные числа, а затем укажите, в каких случаях принимают единичные значения разряды регистра состояния: переноса (C), нулевого результата (Z), отрицательного результата (N).

Пример решения:
```
  10101010
  11111111   Z N C
 110101001   0 1 1
```

а)
```
  00001111
  11110000
```

б)
```
  00111011
    11000101
```

в)
```
  11111111
  11111111
```

г)
```
  00000001
    11111110
```

д)
```
  01010100
  11001100
```

е)
```
  00000001
    01111111
```

ж)
```
  00001111
    00010000
```

з)
```
  11000000
  10000001
```

22. Опишите программу установки разряда нулевого результата в единичное состоя-

ние после троекратного увеличения на 1 содержимого некоторого 8-разрядного регистра. Укажите, чему должно быть равно начальное содержимое этого регистра.

23. Разряд переноса в регистре состояния может являться указателем а) нулевого результата операции, б) отрицательного результата, в) заема или г) всех перечисленных признаков? Какое из этих утверждений справедливо?

3.9 Буферные регистры АЛУ

На рис. 3.1 показаны два буферных регистра, каждый из которых предназначен для временного хранения одного слова данных. Один из этих регистров (ближайший к аккумулятору на рис. 3.1) называется *буфером аккумулятора* АЛУ. О его функциональном назначении речь пойдет несколько позже. Что касается другого буферного регистра, то в него на временное хранение поступают данные с внутренней шины микропроцессора. Необходимость в таком регистре вызвана отсутствием в АЛУ своего запоминающего устройства. В состав АЛУ включены только комбинационные схемы, и поэтому при поступлении исходных данных на входе АЛУ немедленно появляются результирующие данные на его выходе как следствие выполнения операций данной программы.

АЛУ должно получать данные с внутренней шины микропроцессора, модифицировать их, а затем помещать обработанные данные в аккумулятор. Но это неосуществимо без регистра временного хранения данных. Вот почему столь существенна его роль в функционировании микропроцессора. Отметим, что буферные регистры не могут быть использованы программистом.

Если на вход описанного выше буферного регистра могут поступать данные только с внутренней шины данных микропроцессора, то на вход другого буферного регистра, именуемого буфером аккумулятора, данные могут поступать, кроме того, и с выхода аккумулятора. Когда в арифметической или логической операции АЛУ участвуют два слова, одно из них поступает из аккумулятора. Результат подобной операции помещается в аккумулятор. Буфер аккумулятора позволяет избежать ситуации, при которой вход и выход АЛУ одновременно подсоединены к одной и той же точке схемы. Как и другой буферный регистр, буфер аккумулятора недоступен программисту для использования.

Задания для самопроверки

24. Какое основное назначение буферного регистра: а) соединять АЛУ с внутренней шиной данных микропроцессора, б) соединять АЛУ с аккумулятором, в) разделять во времени порты входа и выхода АЛУ или г) служить для аккумулятора запоминающим устройством?

25. Сколько разрядов должен иметь буферный регистр 16-разрядного микропроцессора: а) 32, б) 16, в) 8 или г) 4?

26. Необходимо прибавить слово памяти к младшему байту счетчика команд. Полученный таким образом адрес со смещением должен быть помещен в регистр адреса памяти. Опишите последовательность действий микропроцессора по выполнению этих операций. Можно воспользоваться любым из буферных регистров.

3.10. Регистры общего назначения

Все микропроцессоры имеют шесть описанных выше основных регистров. В дополнение к ним некоторые микропроцессоры располагают другими регистрами, предоставляемыми в распоряжение пользователей. Эти регистры получили название *регистров общего назначения*. В некоторых микропроцессорах они служат в качестве запоминающих устройств, в других функциональные возможности этих регистров не уступают возможностям аккумулятора. Последнее достигается в том случае, если АЛУ может помещать в них данные. Гипотетический микропроцессор, описываемый в данной книге, имеет три регистра общего назначения: B, C и D (см. рис. 3.1). Поскольку в нашем случае АЛУ не помещает данные в эти три регистра, последние не обладают функциональными возможностями аккумулятора. Тем не менее при выполнении многих команд используются эти регистры общего назначения.

Для реализации многих операций использование 8-разрядных регистров B, C и D идентично. Выбор конкретного регистра для выполнения определенного вида работ определяется лишь тем, какой из них доступен и кажется наиболее удобным. Обычно операции, использующие эти регистры, влияют на содержимое регистра состояния. Следовательно, любой из регистров B, C или D можно использовать в качестве счетчика отрицательных приращений. Так, если содержимое используемого для этих целей регистра D становится равным нулю, разряд нулевого результата регистра состояния принимает единичное значение.

Регистры B и C совместно могут выполнять функции 16-разрядного регистра специального назначения. Будем называть их *регистровой парой* BC. Рассматриваемый нами микропроцессор обладает адресацией такого типа, при которой содержимое пары регистров BC загружается в регистр адреса памяти. Это позволяет выполнять регистровые арифметические операции с помощью 16-разрядного регистра. Так, можно задавать приращение содержимому пары регистров BC, а затем использовать это содержимое для адресации памяти.

Следует помнить, что регистры B и C в любое время могут функционировать и как независимые регистры. Таким образом, регистры B и C можно использовать совместно или раздельно, а регистр D всегда выступает в роли отдельного 8-разрядного регистра.

Задания для самопроверки

27. Какую роль могут играть регистры B, C и D: а) счетчика команд, б) регистра адреса памяти, в) регистров общего назначения или г) регистровой пары DC?

28. Регистровая пара BC используется для адресации памяти. Чем это объясняется: а) близостью их местоположения к счетчику команд, б) близостью их нахождения к регистру адреса памяти, в) возможностью использования каждого 8-разрядного регистра самостоятельно или г) потенциальной возможностью адресоваться к любой области памяти посредством 16 бит?

29. Что можно использовать в качестве 16-разрядного счетчика отрицательных приращений: а) регистр D, б) регистр C, в) регистр B или г) регистровую пару BC?

3.11. Схемы управления

Роль схем управления в микропроцессоре чрезвычайно важна и заключается в поддержании требуемой последовательности функционирования всех остальных его звеньев. По «распоряжению» схем управления очередная команда извлекается из регистра команд, определяется, что́ необходимо делать с данными, а затем генерируется последовательность действий по выполнению поставленной задачи.

Обычно работа схем управления микропрограммируется. Это свидетельствует о сходстве архитектуры системы управления микропроцессора с архитектурой некоторого микропроцессора специального назначения. Можно сказать, что схемы управления – это маленький микропроцессор внутри микропроцессора. Одна из главных функций схем управления – декодирование команды, находящейся в регистре команд, посредством дешифратора команд, который в результате выдает сигналы, необходимые для выполнения команды.

На рис. 3.3 показана структурная схема рассматриваемого нами микропроцессора, на которой линии управления изображены черным цветом. Эти линии соединяют схемы управления со всеми узлами микропроцессора, а также с внешними блоками: памяти и ввода-вывода.

Одной из важных входных линий управления, соединяющих микропроцессор с внешними устройствами, является линия связи с генератором тактовых импульсов (таймером), синхронизирующим во времени работу микропроцессора. Принимаемые тактовые сигналы схемы управления преобразуются в многофазные синхросигналы. Рис. 3.4 иллюстрирует указанное преобразование на примере формирования двух синхросигналов с разными фазами из исходных тактовых импульсов. (Иногда микропроцессор использует сформированные подобным образом синхросигналы с четырьмя различными фазами.) В соответствии с рис. 3.4 в течение одного периода тактовых импульсов в микропроцессоре возможны две группы событий: первая – во время действия синхросигнала с фазой ϕ_1, вторая – во время действия синхросигнала

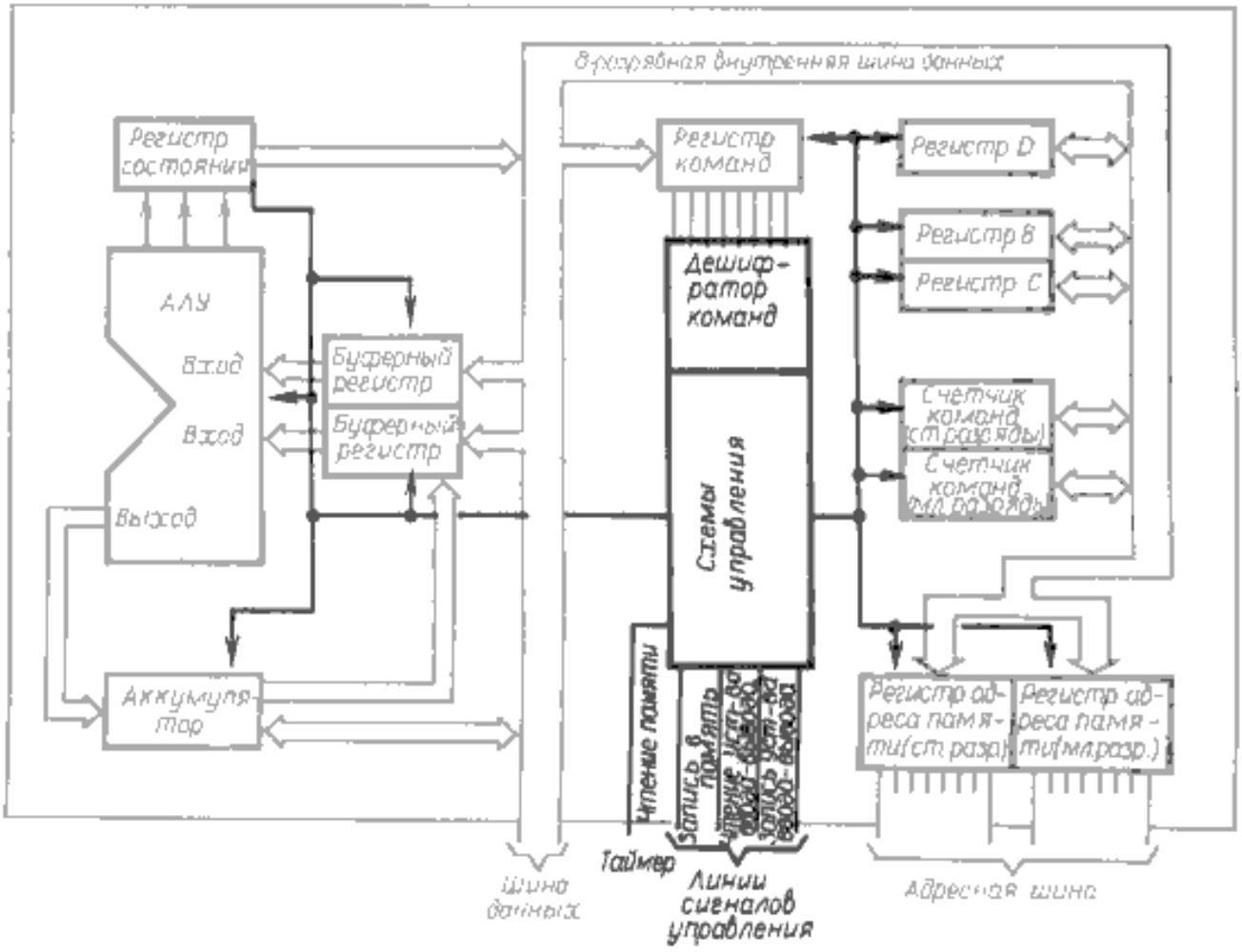


Рис. 3.3. Структурная схема 8-разрядного микропроцессора.

с фазой ϕ_2. Как правило, схемы управления генерируют сигналы в соответствии с той или иной фазой синхросигналов и формируют в эти моменты времени соответствующие выходные сигналы для внешних устройств, таких, как память или устройства ввода-вывода.

В качестве источника тактовых импульсов обычно используется кварцевый генератор – внешний или внутренний, встроенный в микропроцессор.

Помимо указанных выше действий, схемы управления выполняют некоторые другие специальные функции, такие, как управление последовательностью включения питания, управление процессами прерываний. Прерывание – это своего рода запрос, поступающий на схемы управления от других устройств (памяти, ввода-вывода). Прерывание связано с использованием внутренней шины данных микропроцессора. Схемы управления принимают решение, когда и в какой последовательности другие устройства могут пользоваться внутренней шиной данных.

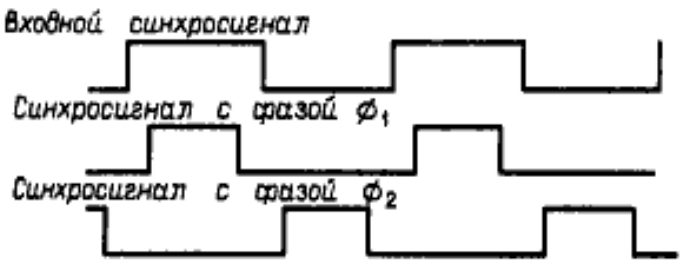


Рис. 3.4. Формирование двух синхросигналов с разными фазами.

3.12. Внутренняя шина данных микропроцессора

Структурная схема микропроцессора (рис. 3.1) показывает, что 8-разрядная внутренняя шина данных соединяет между со-

бой АЛУ и регистры, осуществляя передачу данных внутри микропроцессора. Хотя сигналы управления и играют жизненно важную роль в процессе передачи данных по внутренней шине, тракт их передачи не принадлежит шине данных.

Каждый функциональный блок микропроцессора всегда подключен к внутренней шине данных, однако воспользоваться ею может только после получения соответствующего сигнала от схем управления. Для лучшего понимания работы внутренней шины данных рассмотрим последовательность действий, связанных с добавлением содержимого регистра D к содержимому аккумулятора.

1. В исходном состоянии в аккумуляторе находится двоичное число 11011010, в регистре – число 11011110 (рис. 3.5), причем ни один из этих регистров не соединен ни с каким другим функциональным узлом микропроцессора. В регистре команд содержится код команды ADD (СЛОЖЕНИЕ).

2. Содержимое аккумулятора загружается в соединенный с ним буферный регистр (буфер аккумулятора). Данные, размещенные в регистре D, подаются на внутреннюю шину данных микропроцессора. Другой буферный регистр АЛУ, подключенный к этой шине, загружается копией данных из регистра D (рис. 3.6). В течение этого периода времени только регистр D и буферный регистр используют внутреннюю шину данных. Выходы обоих буферных регистров активируют работу входных портов АЛУ.

3. Инициализируется работа АЛУ по сложению данных, поступивших на его входы. Через выходной порт, подключенный к аккумулятору, результат сложения помещается в последний (рис. 3.7). Следует обратить внимание, что в результате сложения указанных чисел два разряда состояния прини-

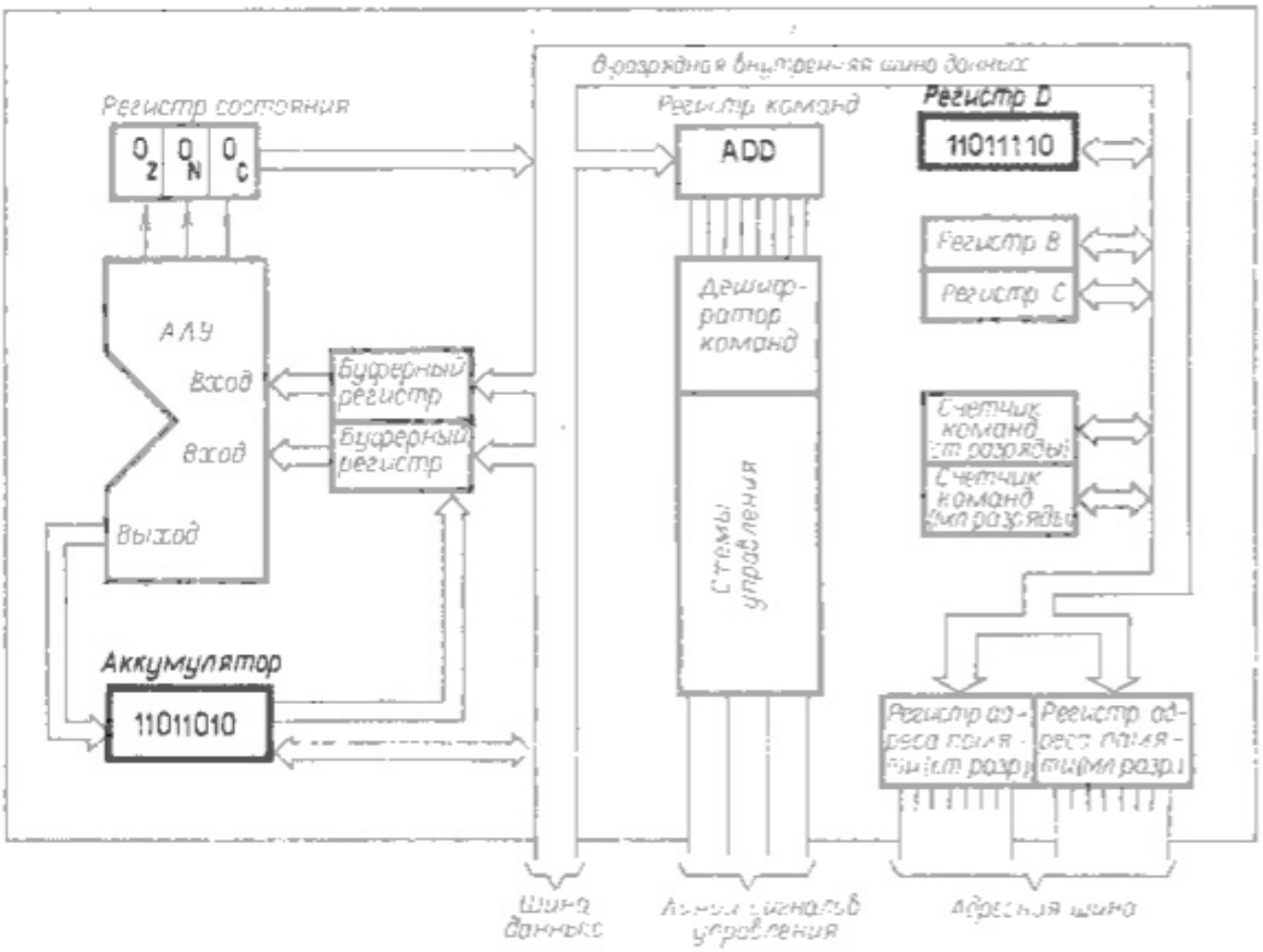


Рис. 3.5. Структурная схема 8-разрядного микропроцессора (аккумулятор и регистр D загружены данными, и регистре команд находится команда ADD; в это время регистр D и аккумулятор не соединены ни с какими другими узлами).

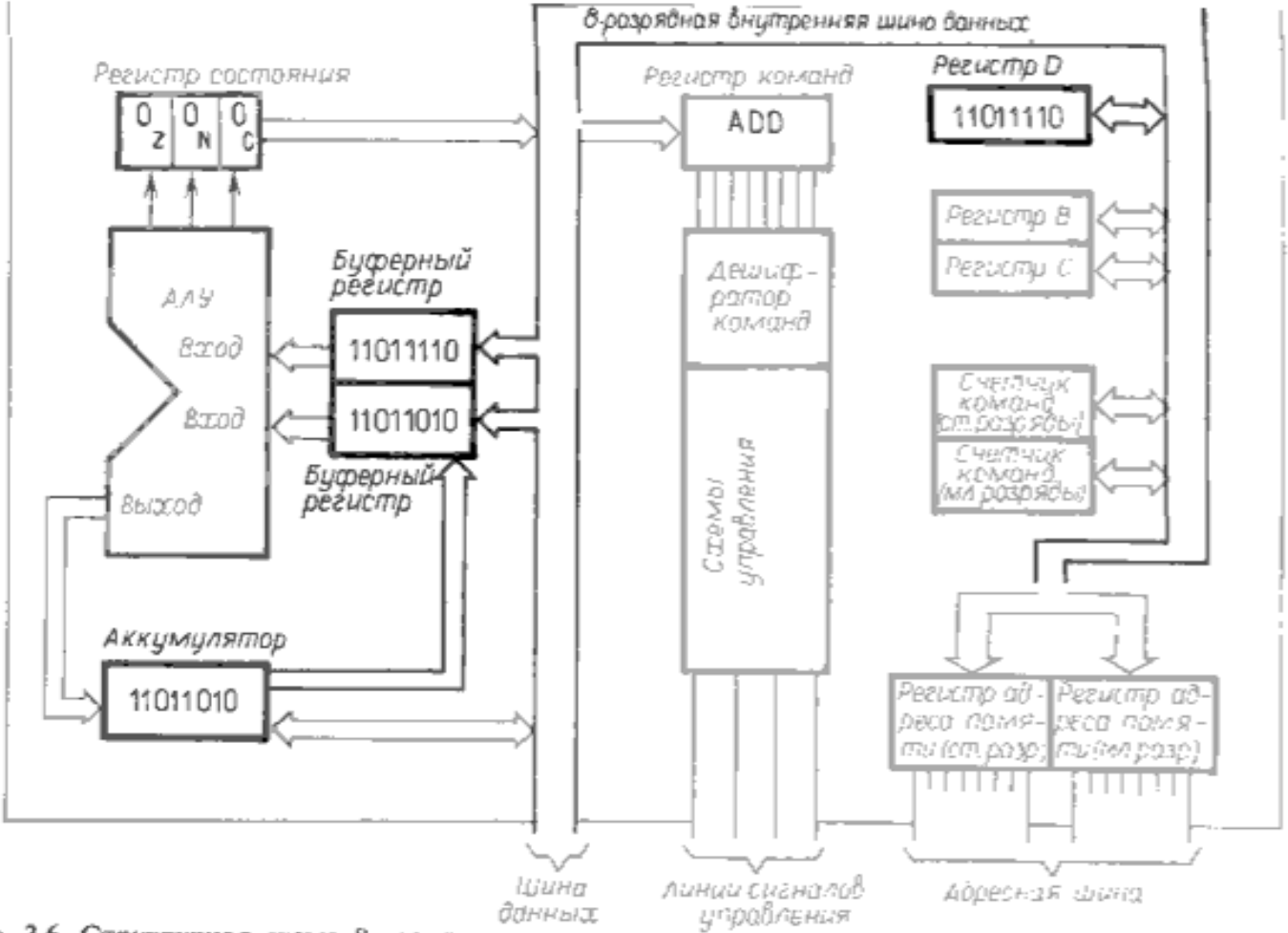


Рис. 3.6. Структурная схема 8-разрядного микропроцессора.

Содержимое аккумулятора пересылается в один из буферных регистров АЛУ; другой буферный регистр АЛУ загружен копией содержимого регистра D, в это время только регистр D и буферный регистр используют внутреннюю шину данных.

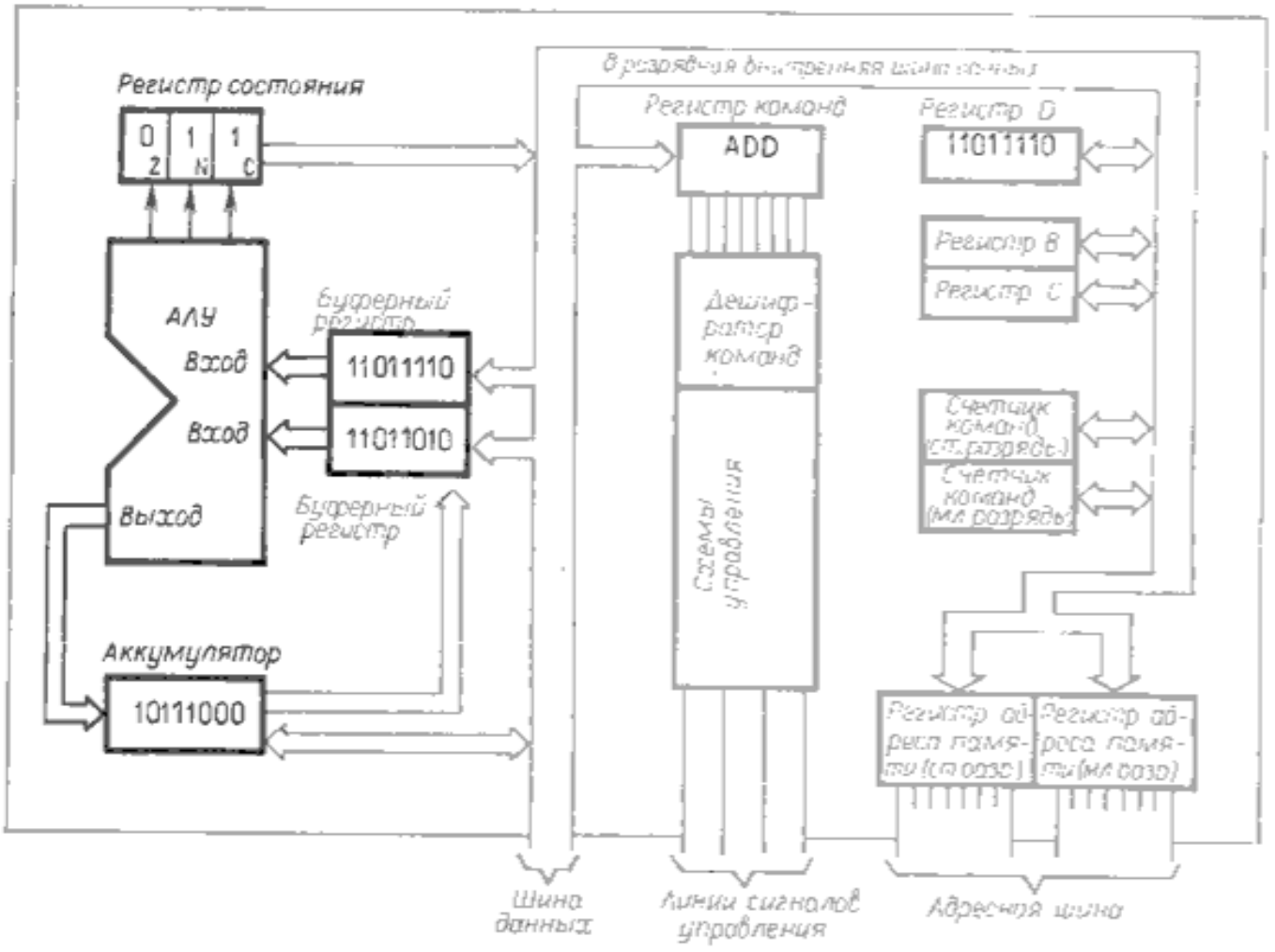


Рис. 3.7. Структурная схема 8-разрядного микропроцессора.

АЛУ получает распоряжение выполнить операцию сложения, с выхода АЛУ информация поступает в аккумулятор. В регистре состояния разряд переноса и знаковый разряд (разряд отрицательного результата) устанавливаются в 1, исходное содержимое аккумулятора оказывается утерянным.

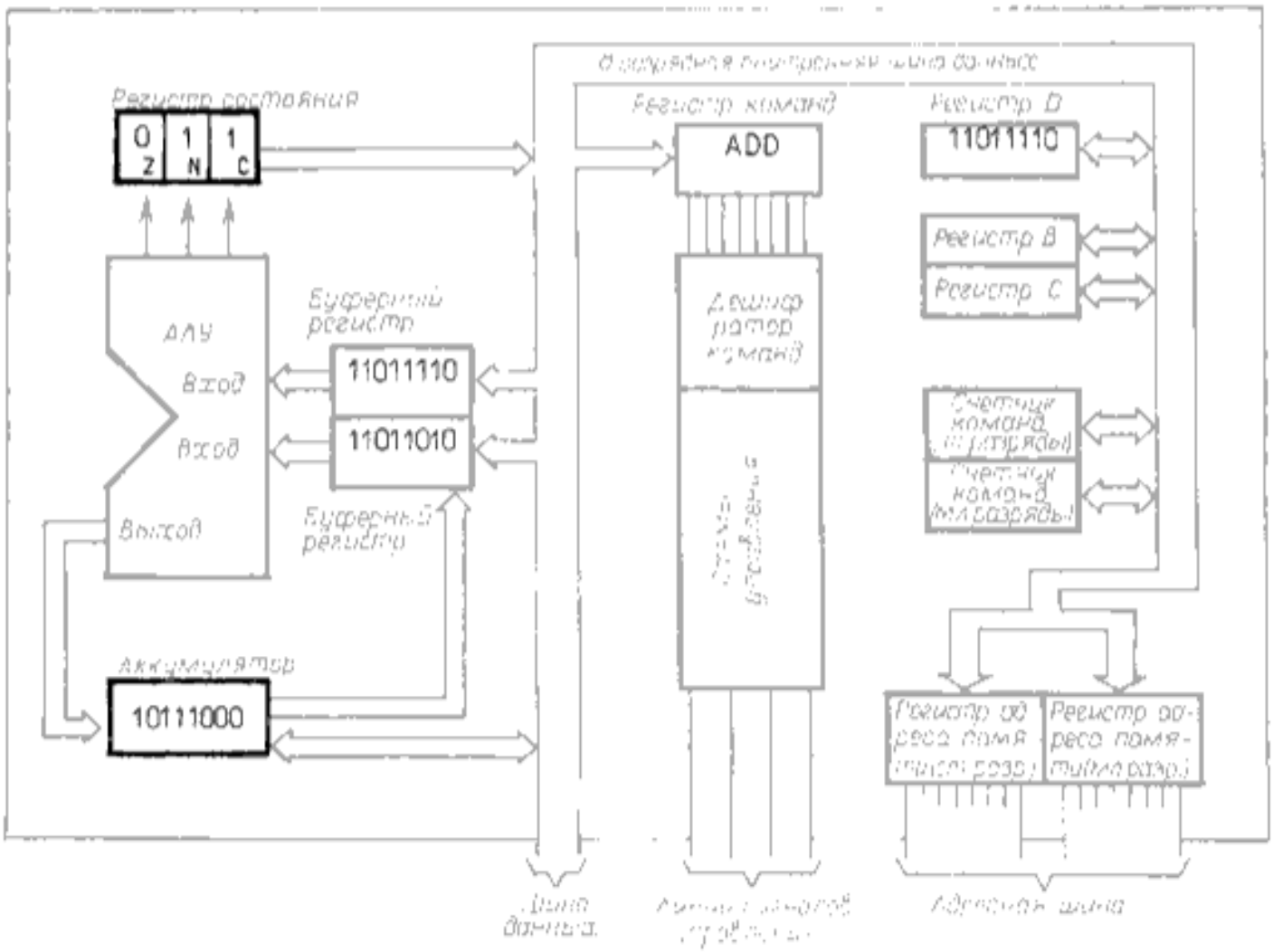


Рис. 3.8. Структурная схема 8-разрядного микропроцессора (операция завершена, в аккумуляторе новые данные, процессор ожидает следующую команду).

мают единичные значения – переноса и знаковый:

```
  1101 1110
+ 1101 1010
-----------
1 1011 1000
```

Перенос Признак отрицательного результата

4. Вход аккумулятора и выходной порт АЛУ отключаются от внутренней шины данных. Результат операции – искомая сумма – находится в аккумуляторе (рис. 3.8). Шина свободна для других операций.

Почти все функциональные узлы микропроцессора имеют двустороннюю связь с внутренней шиной данных, т.е. они могут и посылать данные на шину, и принимать с нее данные. Внутренняя шина данных представляет собой линию двусторонней связи. Следует помнить, что по шине передаются слова данных, а не отдельные биты. Так, в 16-разрядном микропроцессоре все пересылки по шине осуществляются группой из 2 байт (16 бит).

30. Схемы управления направляют пошаговое выполнение микропроцессором той или иной программы. Какие из перечисленных ниже функций можно также отнести к функциям схем управления: а) временное хранение данных, б) управление последовательностью включения питания, в) сложение двух чисел, г) передачу данных?

31. Шина микропроцессора служит для двусторонней связи. Означает ли это, что: а) все данные перемещаются в двух противоположных направлениях, б) данные могут перемещаться в направлении, необходимом

для завершения передачи, в) каждый функциональный узел микропроцессора имеет два входных порта или г) каждый узел должен иметь входной и выходной порты?

32. Все функциональные узлы микропроцессора соединены с внутренней шиной данных. Какой из ниже перечисленных узлов информирует остальные о необходимости передавать данные, принимать их или не выполнять никаких действий: а) аккумулятор, б) схемы управления, в) блок микропроцессора или г) регистр команд?

33. Работа схемы управления синхронизируется высокочастотными импульсными сигналами, называемыми тактовыми импульсами. Каково их назначение: а) информировать о текущем времени, б) управлять работой средств запоминания данных для АЛУ, в) определять временну́ю последовательность функционирования схем управления или г) проверять наличие в регистрах необходимых данных?

Упражнения

3.1. Назовите три основных функциональных узла микропроцессора (см. рис. 3.1).

3.2. Что является связующим звеном для всех функциональных узлов микропроцессора?

3.3. Дайте краткое описание работы АЛУ. Как эта работа влияет на данные в микропроцессоре?

3.4. Назовите четыре специфические функции АЛУ.

3.5. У АЛУ два входных порта. Всегда ли оба порта используются? Почему?

3.6. Каково назначение регистра микропроцессора?

3.7. Что из сказанного ниже о регистрах микропроцессора справедливо: а) регистры соединены с шиной, б) являются элементами памяти объемом в одно слово, в) имеют общее или специальное назначение, г) все перечисленные утверждения?

3.8. Какие из указанных ниже регистров не принадлежат шести основным регистрам микропроцессора: а) регистр адреса памяти, б) индексный регистр, в) аккумулятор, г) счетчик команд?

3.9. Для чего используется аккумулятор: а) для указания адреса области памяти, б) для указания следующей команды, в) для хранения данных с целью выполнения арифметических и логических операций или г) для хранения команды в процессе ее выполнения?

3.10. Куда обычно помещается результат выполнения операции АЛУ: а) в счетчик команд, б) аккумулятор, в) регистр команд или г) в буферный регистр?

3.11. Чему равно число разрядов аккумулятора 12-разрядного микропроцессора: а) 8, б) 12, в) 16 или г) 24?

3.12. Сколько бйтов можно разместить в аккумуляторе двойной длины 12-разрядного микропроцессора: а) 8, б) 12, в) 16 или г) 24?

3.13. Зачем может понадобиться аккумулятор двойной длины?

3.14. Некоторые микропроцессоры располагают двумя аккумуляторами, что обеспечивает им бóльшую гибкость функционирования. Приведите пример, подтверждающий это.

3.15. Счетчик команд указывает, какая команда подлежит выполнению. Что можно сказать об очередности выполнения этой команды?

3.16. Какое количество разрядов должно быть у счетчика команд 8-разрядного микропроцессора, чтобы адресоваться к 65 536 различным областям памяти: а) 4, б) 8, в) 16 или г) 24?

3.17. Почему счетчик команд должен обеспечивать адресацию любой области памяти, даже если программа не использует все области?

3.18. При включении питания микропроцессора команда начальной установки задает счетчику команд адрес некоторой области памяти. Какой из приведенных здесь ответов верен: а) одной и той же заранее определенной области, б) области 0000, в) области FFFF или г) случайно выбранной области?

3.19. Какой шиной управляет выход регистра адреса памяти?

3.20. Для какой цели используются данные, поступающие по адресной шине?

3.21. Чему равно число разрядов регистра адреса памяти 16-разрядного микропроцессора, если необходимо адресоваться к 1 048 576 областям памяти?

3.22. Регистр команд хранит команду, пока она выполняется. Куда подсоединены вход и выход этого регистра?

3.23. В течение двухфазного цикла работы микропроцессор извлекает команду из памяти и выполняет. Как называется такой цикл?

3.24. В течение какого цикла работы микропроцессора команда извлекается из памяти и помещается в регистр команд?

3.25. В течение какого цикла команда декодируется и выполняется?

3.26. Чему равно количество битов содержимого регистра команд: а) 4, б) 8, в) 16 или г) это число не фиксировано?

3.27. Назовите три обычно используемых бита регистра состояния. Кратко опишите условия присвоения им единичных значений.

3.28. Содержимое регистра состояния проверяется некоторыми командами программы. Для

какой цели используются результаты такой проверки?

3.29. Объясните назначение буферного регистра микропроцессора. Следует ли ожидать расположение буферного регистра на том же месте для различных микропроцессоров? Почему?

3.30. Чем обеспечивает таймер микропроцессора схемы управления последнего: а) регистром, б) установкой начального состояния, в) тактовыми синхросигналами или г) командой?

3.31. Каково назначение схем управления микропроцессора?

3.32. Является ли внутренняя шина данных микропроцессора а) двунаправленной, б) равной по разрядности слову, в) используемой для взаимного соединения функциональных узлов микропроцессора?

3.33. Назовите функциональный узел микропроцессора, имеющий двустороннюю связь с внутренней шиной данных. Назовите узел, не имеющий такой связи.

Ответы на вопросы заданий для самопроверки

1. в. 2. в. 3. б. 4. в. 5. а. 6. в. 7. а. 8. г. 9. г. 10. в. 11. а. 12. б. 13. в. 14. б. 15. в. 16. г. 17. в. 18. б. 19. а. 20. г.

21.

	Результат	Z N C
а)	1111 1111	0 1 0
б)	1 0000 0000	1 0 1
в)	1 1111 1110	0 1 1
г)	1111 1111	0 1 0
д)	1 0010 0000	0 0 1
е)	1000 0000	0 1 0
ж)	0001 1111	0 0 0
з)	1 0100 0001	0 0 1

22. Необходимо выполнить следующие три шага:

Исходное состояние	1111 1101
Первое приращение	1111 1110
Второе приращение	1111 1111
Третье приращение	0000 0000

В результате выполнения третьего шага разряду нулевого результата присваивается единичное значение.

Можно было бы выполнить три последовательных шага для другого исходного состояния, а именно:

Исходное состояние	0111 1101
Первое приращение	0111 1110
Второе приращение	0111 1111
Третье приращение	1000 0000

В этом случае в регистре состояния проверке подлежит разряд отрицательного результата.

23. в. 24. в. 25. б.

26. Одна из возможных последовательностей шагов имеет следующий вид:

а) Загрузка слова смещения в аккумулятор.

б) Загрузка содержимого аккумулятора в буферный регистр.

в) Загрузка младшего байта счетчика команд в буферный регистр АЛУ.

г) Запись результата (суммы) в аккумулятор.

д) Загрузка старшего байта счетчика команд на место старшего байта регистра адреса памяти.

е) Запись содержимого аккумулятора на место младшего байта регистра адреса памяти.

27. в. 28. г. 29. г. 30. б. 31. б. 32. б. 33. в.

Глава 4.
Двоичная арифметика

В этой главе рассматривается выполнение микропроцессором арифметических операций с двоичными числами. Хотя при составлении программы для микропроцессора можно пользоваться десятичной арифметикой, выполнению программы должно предшествовать преобразование десятичных чисел в двоичные.

В начале разделов кратко описываются правила сложения и вычитания десятичных чисел, поскольку двоичная арифметика во многом подобна десятичной. Далее объясняется, как в микропроцессоре представляются положительные и отрицательные двоичные числа, что такое дополнительный код двоичного числа и почему его использование упрощает манипулирование двоичными числами со знаком, как складываются и вычитаются двоичные числа в дополнительном коде.

Затем рассматриваются двоичные умножение и деление как последовательные повторения операций сложения и вычитания соответственно, демонстрируются правила двоичного деления общего назначения.

В заключение обсуждается вопрос о выполнении арифметических операций с повышенной точностью, позволяющих манипулировать двоичными числами длиной более одного слова. Здесь же описывается арифметика чисел с плавающей точкой, ориентированная на работу с очень большими или очень малыми числами.

4.1. Двоичное сложение

Сложение двоичных чисел подобно сложению десятичных. В обоих случаях операции начинаются с обработки наименьших значащих цифр, расположенных в крайней справа позиции. Если результат сложения наименьших значащих цифр двух слагаемых не помещается в соответствующем разряде результата, то происходит перенос. Цифра, переносимая в соседний слева разряд, добавляется к содержимому последнего. Сложение цифр любых одноименных разрядов может повлечь за собой перенос в более старший разряд. Перенос возникает, если результат сложения цифр одноименных разрядов больше 9 при использовании десятичной арифметики, и больше 1 в случае двоичной системы.

Сходство и различие операций десятичного и двоичного сложения можно продемонстрировать на следующем примере:

	Десятичная арифметика	Двоичная арифметика
Перенос (единицы)	11	1111 1110
Слагаемое	099	0110 0011
Слагаемое	095	0101 1111
Сумма	194	1100 0010

Сложение десятичных чисел 99 и 95 начинается с операции сложения над разрядом единиц: 9 + 5 = 14. В разряд единиц резуль-

тата записывается число 4, а оставшаяся 1 переносится в соседний слева разряд десятков. Затем производится сложение значения разряда десятков обоих слагаемых с добавлением 1, «перенесенной» из разряда единиц: $9 + 9 + 1 = 19$. В разряд десятков записывается число 9, а 1 переносится в разряд сотен. Процедура сложения закончена, искомая сумма равна 194.

Сложение десятичных чисел (например, 99 и 95) существенно упрощается, если воспользоваться так называемой таблицей сложения (табл. 4.1, *а*). Результат сложения любых двух цифр находится на пересечении соответствующих строки и столбца. Если это пересечение находится в той части таблицы, которая расположена на темном фоне, то сложение данных цифр порождает перенос 1 в ближайший старший разряд. Подобная процедура сложения десятичных чисел столь привычна для нас, что обычно мы выполняем ее автоматически.

Таблица 4.1

Таблицы сложения

а) Десятичное сложение

+	Слагаемое									
	0	1	2	3	4	5	6	7	8	9
Слагаемое 0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

б) Двоичное сложение

+	Слагаемое	
	0	1
Слагаемое 0	0	1
1	1	0

Двоичная арифметика вначале может показаться менее удобной. Однако в действительности двоичная арифметика намного проще десятичной, особенно если прибегнуть к табл. 4.1, *б*. Результат сложения двоичных цифр одноименных разрядов находится на пересечении соответствующих строки и столбца. Если пересечение происходит на темном фоне, то имеет место перенос единицы в ближайший старший разряд. Сравнение таблиц для случаев двоичного и десятичного сложения показывает, что процедура сложения двоичных чисел намного проще.

Вернемся еще раз к приведенному выше примеру и рассмотрим сложение двоичных эквивалентов десятичных чисел 99 и 95, т. е. 0110 0011 и 0101 1111. Начнем с наименьшего по значимости разряда слагаемых. В данном случае они представляют собой двоичные единицы, сложение которых, согласно табл. 4.1, *б*, приводит к появлению нуля в данном разряде и переносу единицы в следующий разряд. В ближайшем к крайнему справа разряду результат получается путем сложения трех двоичных единиц: $1 + 1 + 1 = 11$. Эту процедуру можно представить как две последовательные операции сложения: $1 + 1 = 10$ и $10 + 1 = 11$. Двоичное сложение пары чисел выполняется согласно правилам, указанным в табл. 4.1, *б*. Сложение двух единиц дает нуль в самом младшем разряде результата и перенос единицы в следующий, более старший разряд. Добавление единицы к нулю младшего разряда не приводит к переносу единицы в следующий разряд. Окончательный результат равен 11, это — двоичное выражение разряда «двоек» искомой суммы. В этом разряде размещается «правая» единица результата, а «левая» переносится в следующий по старшинству разряд, т. е. разряд «четверок», где она складывается с нулем и единицей соответствующих разрядов слагаемых. В результате в разряде «четверок» образуется нуль, а единица переносится в разряд «восьмерок». Здесь эта единица складывается с нулем и единицей – значением одноименного разряда слагаемых. Как следствие, в разряд «восьмерок» результата записывается нуль, а в разряд «шестнадцати» переносится единица, которая подле-

жит сложению с нулем и единицей разряда «шестнадцати» слагаемых.

После выполнения операции сложения в разряде «шестнадцати» результата формируется нуль с переносом единицы в разряд «тридцати двух». Здесь эта единица складывается с единицей и нулем слагаемых, приводя к появлению нуля в разряде «тридцати двух» результата и переносу единицы в разряд «шестидесяти четырех». Единица переноса складывается с двумя единицами слагаемых, что обусловливает единицу в результате данного разряда и перенос единицы в следующий по старшинству разряд. В этом самом старшем разряде «ста двадцати восьми» единица переноса складывается с двумя нулями соответствующего разряда слагаемых, формируя единицу. Итак, сумма двоичных чисел 0110 0011 и 0101 1111 равна 1100 0010.

Рассмотренный пример свидетельствует о простоте процедуры двоичного сложения. Единственное неудобство, присущее двоичным операциям,– громоздкость записи больших чисел в двоичной форме, что вызывает множество переносов из одного разряда в другой при выполнении операции сложения.

В показанном здесь примере использованы 8-разрядные двоичные числа, так как большинство микропроцессоров оперирует 8-разрядными словами. Хотя для представления слагаемых можно ограничиться меньшим количеством разрядов, по указанной выше причине приведенные в данном примере двоичные числа имеют «ведущие» (незначащие) нули. Оперируя десятичными числами, мы обычно стараемся избежать записи ведущих нулей. Однако в упомянутом примере сложение двух чисел приводит к переносу единицы в разряд сотен, поэтому слагаемые содержат по одному незначащему нулю перед первой значащей цифрой.

Задания для самопроверки

1. Сложите приведенные ниже двоичные числа:

а) $\begin{array}{r} 0000\,0101 \\ +\ 0001\,0001 \\ \hline \end{array}$ б) $\begin{array}{r} 1000\,1001 \\ +\ 0000\,1111 \\ \hline \end{array}$ в) $\begin{array}{r} 0111\,1111 \\ +\ 0111\,1110 \\ \hline \end{array}$

г) $\begin{array}{r} 0101\,0101 \\ +\ 1010\,1010 \\ \hline \end{array}$ д) $\begin{array}{r} 101 \\ +\ 011 \\ \hline \end{array}$ е) $\begin{array}{r} 1001 \\ +\ 011 \\ \hline \end{array}$

ж) $\begin{array}{r} 0100\,0010\,0110\,1100 \\ +\ 0101\,1110\,1001\,0110 \\ \hline \end{array}$

з) $\begin{array}{r} 0111\,1111\,1111\,1111 \\ +\ 0001\,0111\,1011\,1001 \\ \hline \end{array}$

2. Представьте указанные ниже десятичные слагаемые их 8- или 16-разрядными двоичными эквивалентами и выполните операции сложения:

а) $\begin{array}{r} 101 \\ -\ 16 \\ \hline \end{array}$ б) $\begin{array}{r} 225 \\ -\ 168 \\ \hline \end{array}$ в) $\begin{array}{r} 398 \\ -\ 132 \\ \hline \end{array}$

г) $\begin{array}{r} 56 \\ -\ 10 \\ \hline \end{array}$ д) $\begin{array}{r} 86 \\ -\ 25 \\ \hline \end{array}$ е) $\begin{array}{r} 289 \\ -\ 493 \\ \hline \end{array}$

3. Какое максимальное количество разрядов необходимо для представления результата сложения двух 8-разрядных двоичных чисел?

4. Чем отличаются двоичные числа 101, 0101 и 0000 0101? Как это может повлиять на результат двоичного суммирования?

4.2. Двоичное вычитание

Двоичное вычитание подобно десятичному вычитанию. Как и в случае сложения, различие выполнения вычитания в двоичной и десятичной форме состоит лишь в особенностях поразрядных операций. Реализация последних существенно облегчается при использовании таблиц вычитания. Табл. 4.2, *а* предназначена для десятичного вычитания. Если содержимое разряда уменьшаемого меньше содержимого одноименного разряда вычитаемого, то результат – пересечение соответствующей строки и столбца – следует искать в той части таблицы, которая расположена на темном фоне. Это означает заем единицы из соседнего старшего разряда.

Из табл. 4.2, *б* следует, что сказанное справедливо и для двоичного вычитания, т. е., если уменьшаемое меньше вычитаемого, имеет место заем. А это происходит в том

Таблица 4.2

Таблицы вычитания

а) Десятичное вычитание

—	Вычитаемое									
	0	1	2	3	4	5	6	7	8	9
Уменьшаемое 0	0	9	8	7	6	5	4	3	2	1
1	1	0	9	8	7	6	5	4	3	2
2	2	1	0	9	8	7	6	5	4	3
3	3	2	1	0	9	8	7	6	5	4
4	4	3	2	1	0	9	8	7	6	5
5	5	4	3	2	1	0	9	8	7	6
6	6	5	4	3	2	1	0	9	8	7
7	7	6	5	4	3	2	1	0	9	8
8	8	7	6	5	4	3	2	1	0	9
9	9	8	7	6	5	4	3	2	1	0

б) Двоичное вычитание

—	Вычитаемое	
	0	1
Уменьшаемое 0	0	1
1	1	0

случае, когда из нуля, содержащегося в двоичном разряде, вычитается единица. Благодаря заему единицы из ближайшего старшего разряда в одноименном разряде результата получается единица.

Сравнение процедур десятичного и двоичного вычитания можно продемонстрировать на следующем примере:

	Десятичная арифметика	Двоичная арифметика
Заем (единица)	1	1100000
Уменьшаемое	109	01101101
Вычитаемое	49	00110001
Разность	060	00111100

Десятичное вычитание начинается операцией над содержимым самых младших (крайних справа) разрядов единиц уменьшаемого и вычитаемого: $9 - 9 = 0$, причем заем из старшего разряда уменьшаемого не требуется. Затем подобная операция повторяется для разряда десятков. Однако в данном случае требуется заем единицы из разряда сотен уменьшаемого, в результате чего там остается нуль. После выполнения вычитания для разряда десятков значение одноименного разряда результата равно числу 6 ($10 - 4 = 6$). Наконец, операция вычитания выполняется для разряда сотен, приводя к нулю в одноименном разряде результата ($0 - 0 = 0$).

Двоичное вычитание также начинается операцией над значением крайних справа двоичных разрядов уменьшаемого и вычитаемого: $1 - 1 = 0$. Заем в данном случае не требуется. Далее операция повторяется применительно к соседнему разряду: $0 - 0 = 0$, заем из старшего разряда отсутствует. Не требуется заем и при выполнении двух следующих операций вычитания (для третьего и четвертого разрядов): $1 - 0 = 1$. Однако при вычитании значения пятого разряда вычитаемого (1) из значения одноименного разряда уменьшаемого (0) требуется заем единицы из шестого разряда уменьшаемого, при этом результат вычитания равен 1. Произведенный заем приводит к изменению значения шестого разряда уменьшаемого (единица заменяется нулем). В связи с этим при выполнении вычитания для значений шестого разряда ($0 - 1$) необходим заем единицы из седьмого разряда уменьшаемого, в результате чего указанный разряд принимает нулевое значение. После выполнения перечисленных действий значение шестого разряда результата равно 1. Вычитание значения седьмого разряда вычитаемого (0) из значения одноименного разряда уменьшаемого (0) не требует заема, в разряде результата получается 0. То же самое происходит после применения описанной процедуры для восьмого разряда. (Хотя в последнем разряде находятся незначащие нули, необходимо выполнить операцию вычитания и в этом случае, поскольку мы оперируем 8-разрядными двоичными числами.)

Итак, процедура двоичного вычитания относительно проста. Для ее освоения требуется лишь некоторая практика.

Задания для самопроверки

5. Произведите операцию вычитания над приводимыми ниже парами двоичных чисел:

а) $\begin{array}{r} 0101 \\ -\ 0001 \\ \hline \end{array}$ б) $\begin{array}{r} 1001 \\ -\ 111 \\ \hline \end{array}$ в) $\begin{array}{r} 1111 \\ -\ 1001 \\ \hline \end{array}$

г) $\begin{array}{r} 1001 \\ -\ 0001 \\ \hline \end{array}$ д) $\begin{array}{r} 1001\,0110 \\ -\ 0110\,1001 \\ \hline \end{array}$ е) $\begin{array}{r} 1111\,0001 \\ -\ 0000\,0011 \\ \hline \end{array}$

ж) $\begin{array}{r} 1001\,0011\,1001\,1001 \\ -\ 0101\,1101\,1111\,0000 \\ \hline \end{array}$

6. Представьте приводимые ниже десятичные числа их 4-, 8- или 16-разрядными двоичными эквивалентами и выполните соответствующие операции вычитания:

а) $\begin{array}{r} 15 \\ -\ 12 \\ \hline \end{array}$ б) $\begin{array}{r} 8 \\ -\ 2 \\ \hline \end{array}$ в) $\begin{array}{r} 255 \\ -\ 24 \\ \hline \end{array}$

г) $\begin{array}{r} 52 \\ -\ 36 \\ \hline \end{array}$ д) $\begin{array}{r} 29 \\ -\ 12 \\ \hline \end{array}$ е) $\begin{array}{r} 136 \\ -\ 108 \\ \hline \end{array}$

7. По какой причине происходит заем из более старшего разряда числа?

8. В чем разница между переносом в более старший разряд и заемом из него?

4.3. Двоичные числа в дополнительном коде

При первоначальном знакомстве с двоичными числами складывается впечатление, что двоичная система счисления позволяет записывать только целые положительные числа. Однако мы уже знаем, что использование двоичной точки – это обычное явление, а следовательно, возможно представление дробных двоичных чисел. Возникает вопрос: можно ли записать отрицательные двоичные числа, и если можно, то каким образом?

Существует несколько способов представления отрицательных двоичных чисел. Большинство из них не соответствует возможностям «двоичной электроники», т. е. аппаратной основе АЛУ. Однако анализ этих способов и требований, связанных с работой электронных схем, привел к разработке приемлемых решений. Одно из них известно как *представление числа посредством величины и знака*, причем бит знака занимает самый старший разряд поля представления двоичного числа. Если число положительное, бит знака равен 0, если оно отрицательное, то этот бит равен 1. Рассмотрим, например, десятичное число 28. Его 7-разрядный двоичный эквивалент имеет вид 0011100. Если десятичное число положительное (+ 28), то к указанному двоичному эквиваленту следует приписать слева 0 (бит положительного знака), а именно 0001 1100. Если же десятичное число отрицательное (– 28), то требуется добавить 1 (бит отрицательного знака): 1001 1100.

Бит знака, равный нулю для положительных чисел и единице для отрицательных, используется и при записи двоичных чисел в *обратном коде* (в виде так называемого дополнения до 1). Обратный код двоичного отрицательного числа формируется заменой всех нулей числа на единицы, а всех единиц – на нули. Это же выполняется и для самого старшего разряда поля представления числа. Поскольку этот разряд «не занят» битами величины числа, то в исходном состоянии там нуль, а по завершении формирования обратного кода отрицательного числа – единица, выполняющая роль кода знака. Правило формирования обратного кода простое, однако работа с обратными кодами вызывает ряд затруднений. Так, нулевой результат может быть представлен комбинацией или двоичных нулей, или двоичных единиц. Приведем пример записи отрицательного числа в обратном коде: десятичному числу – 28 соответствует двоичная запись 1110 0011. (Сравните с двоичным числом 0001 1100, являющимся эквивалентом десятичного числа + 28.)

В микро-ЭВМ широко используется представление отрицательных чисел в *дополнительном коде* (в виде так называемого дополнения до 2). При таком представлении исчезает двусмысленность представления нулевого результата, присущая записи в обратном коде. Формирование дополнительного кода, или сокращенно дополнения, состоит из двух операций: получения обратного кода и добавления единицы. Как следует из табл. 4.3, это позволяет, например,

Таблица 4.3. **Десятичные эквиваленты двоичных чисел**

8-разрядное двоичное число	Десятичный эквивалент	
	двоичного числа со знаком (отрицательное число в дополнительном коде)	двоичного числа без знака
0000 0000	+ 0	0
0000 0001	+ 1	1
0000 0010	+ 2	2
0000 0011	+ 3	3
.	.	.
.	.	.
.	.	.
0111 1100	+ 124	124
0111 1101	+ 125	125
0111 1110	+ 126	126
0111 1111	+ 127	127
1000 0000	− 128	128
1000 0001	− 127	129
1000 0010	− 126	130
1000 0011	− 125	131
.	.	.
.	.	.
.	.	.
1111 1100	− 4	252
1111 1101	− 3	253
1111 1110	− 2	254
1111 1111	− 1	255

посредством 8 бит представить в двоичной форме десятичные числа от − 128 до + 127, включая 0. В этой таблице показаны два типичных для микропроцессоров способа использования двоичных кодов: как двоичных чисел со знаком, так и без знака. Левый столбец содержит двоичные числа (двоичные коды) от 0000 0000 до 1111 1111, правый столбец – их десятичные эквиваленты от 0 до 255, полученные в предположении, что рассматриваются числа без знака. В центральном столбце находятся десятичные эквиваленты двоичных чисел левого столбца, полученные в предположении, что отрицательные числа записывались в дополнительном коде. Здесь положительным двоичным числам (от 0000 0000 до 0111 1111) соответствуют десятичные числа от 0 до + 127, а содержащим 1 в восьмом разряде отрицательным двоичным числам (от 1000 0000 до 1111 1111) – десятичные от − 128 до − 1.

Используя восемь двоичных разрядов и представляя отрицательные числа в дополнительном коде, можно записать 256 различных чисел: 127 положительных, нуль и 128 отрицательных. Упомянутую выше процедуру формирования дополнения — представление отрицательного числа в обратном коде и добавление единицы — продемонстрируем на следующем примере:

Число 4_{10} в двоичной форме	0000 100
Обратный код числа 4_{10}	1111 011
Добавляемая к обратному коду 1_2	1
Число 4_{10} в дополнительном коде	111 1100

Сопоставьте полученный результат с соответствующим кодом в табл. 4.3.

Для представления двоичного числа в дополнительном коде можно пользоваться другим способом, отличным от описанного выше и более коротким по числу операций. В поисках первого бита, равного единице, просматривают справа налево разряды числа, начиная с наименьшего по значимости. До тех пор пока встречаются нули, их копируют в одноименные разряды результата. Первая встретившаяся единица также копируется в соответствующий разряд результата, но каждый последующий бит исходного числа заменяется на обратный.

Согласно табл. 4.3, арифметические операции над двоичными числами без знака ничем не отличаются от подобных операций над двоичными числами со знаком, отрицательные из которых представлены своими дополнениями. Это существенно упрощает аппаратную реализацию подобных операций в микропроцессоре. Однако следует не упускать из виду, с какими числами вы имеете дело в данный момент: без знака или со знаком. Например, при сложении двух чисел без знака результат – число без знака в виде некоторой последовательности битов, которую можно интерпретировать и как отрицательное число в дополнительном коде.

В общем случае при сложении или вычитании чисел со знаком результат есть число со знаком; если при этом бит старшего разряда равен единице, то результат – отрицательное число в дополнительном коде. Если требуется определить абсолютное значение (величину) результата, последний необходимо представить в обратном коде, а затем прибавить единицу.

В простоте операций над отрицательными

числами в виде дополнений можно убедиться на примере вычитания, которое выполняется следующим образом: определяется дополнительный код вычитаемого и производится сложение этого кода с уменьшаемым. Если разность – число положительное (бит старшего разряда равен 0), то бит переноса необходимо отбросить; полученная последовательность битов и есть двоичный код результата. Если разность – число отрицательное (бит старшего разряда равен 1), то она представлена в дополнительном коде. Выше указывалось, что для определения абсолютной величины отрицательного числа, представленного в таком виде, необходимо применить к нему операцию вычисления дополнительного кода.

Проиллюстрируем операции вычитания следующими примерами.

Пример 1. Вычислить разность чисел 58 – – 23:

а) Определение дополнительного кода числа 23

0001 0111	Число 23_{10}
1110 1000	Обратный код числа 23_{10}
0000 0001	Единица, добавляемая к обратному коду
1110 1001	Дополнительный код числа 23_{10}

б) Вычисление разности

Десятичная арифметика		Двоичная арифметика	
58		0011 1010	Число 58_{10}
–	+		
23		1110 1001	Дополнительный код числа 23_{10}
35	↓	0010 0011	Разность 35_{10}

Единица переноса, отбрасываемая в случае положительного результата

Пример 2. Вычислить разность чисел 26–34:

а) Определение дополнительного кода числа 34

0010 0010	Число 34_{10}
1101 1101	Обратный код числа 34_{10}
0000 0001	Единица, добавляемая к обратному коду
1101 1110	Дополнительный код числа 34_{10}

б) Вычисление разности

Десятичная арифметика		Двоичная арифметика	
26		0001 1010	Число 26_{10}
–	+		
34		1101 1110	Дополнительный код числа 34_{10}
– 08		1111 1000	Разность в форме дополнения (поскольку в старшем разряде 1)

в) Определение абсолютного значения разности

1111 1000	Дополнительный код разности
0000 0111	Обратный код
0000 0001	Единица, добавляемая к обратному коду
0000 1000	Абсолютное значение разности (8_{10})

Задания для самопроверки

9. В какой форме чаще всего представляют отрицательные двоичные числа: а) в обратном коде, б) в дополнительном коде, в) в виде знака и величины или г) в виде абсолютного значения?

10. Определите дополнительный код следующих двоичных чисел:

а) 1011 0110 б) 0110 1011 в) 101 г) 0000 1100 д) 1 е) 0000 0001 ж) 1111 1111 з) 0101 0101 и) 1111 0000 1111 0000 к) 1100 1100 1100 0011

11. Исходя из предположения, что в п. 10 указаны либо 8-, либо 16-разрядные числа в дополнительном коде, определите, какие из них положительные, а какие отрицательные.

12. Приводимые ниже десятичные числа представьте как 8-разрядные числа в дополнительном коде:

а) 64; б) – 56; в) 12; г) 0; д) – 128; е) 127; ж) 32; з) – 32; и) 256; к) 16; л) – 100; м) – 4.

13. Почему с помощью одних и тех же логических схем можно выполнять сложение или вычитание двоичных чисел без знака и дополнительных кодов отрицательных чисел? Приведите пример, подтверждающий ответ.

4.4. Двоичное умножение

Двоичное и десятичное умножение, так же, как и двоичное и десятичное сложение или вычитание, во многом похожи. Умножение – это быстрый способ сложения нескольких одинаковых чисел. Например, умножение 7 на 5 сводится к сложению пяти одинаковых чисел, каждое из которых равно 7.

При умножении одного числа на другое одно из чисел называется *множимым*, другое – *множителем*. Умножение выполняется поразрядно. Часто возникает необходимость переноса в следующий по старшинству разряд. Перемножая десятичные числа, мы обычно «решаем в уме» возникающие при этом проблемы переноса. По завершении умножения множимого на значение младшего разряда множителя получается первое *частичное произведение*. В результате умножения множимого на значение следующего по старшинству разряда множителя формируется второе частичное произведение. Подобная процедура повторяется с целью получения всех необходимых частичных произведений. Поскольку каждое очередное частичное произведение – результат перемножения множимого и разряда множителя, значимость которого в 10 раз больше значимости разряда, использованного в предыдущей операции умножения, то все цифры полученного произведения сдвигаются влево на одну десятичную позицию (разряд). Для получения результирующего произведения смещенные относительно друг друга частичные произведения складываются. Возникающие при сложении переносы должны быть учтены при формировании окончательного результата.

Рассмотрим в качестве примера умножение числа 17 на число 12:

```
  17      Множимое
×
  12      Множитель
----
  34      Первое частичное произведение
 17       Второе частичное произведение
 100      Перенос
----
 204      Результирующее произведение
```

Поскольку множитель состоит из двух разрядов, получаются два частичных произведения, сложение которых вызывает перенос в разряд сотен.

Теперь перейдем к рассмотрению двоичного умножения, схематически представленного с помощью таблицы двоичного умножения (табл. 4.4). Аналогичную таблицу десятичного умножения, состоящую из 10

Таблица 4.4. **Таблица двоичного умножения**

×		Множимое	
		0	1
Множитель	0	0	0
	1	0	1

строк и 10 столбцов, мы знаем наизусть. По сравнению с последней таблицей таблица двоичного умножения чрезвычайно проста: в умножении участвуют цифры, принимающие только два значения – 0 или 1, в результате умножения перенос не возникает никогда.

Воспользуемся таблицей двоичного умножения для вычисления произведения 8-разрядных двоичных эквивалентов десятичных чисел 17 и 12:

```
  00010001  Множимое 17(10)
  00001100  Множитель 12(10)
----------
  00000000  Первое частичное произведение
 00000000   Второе частичное произведение
00010001    Третье частичное произведение
00010001    Четвертое частичное произведение
00000000    Пятое частичное произведение
00000000    Шестое частичное произведение
00000000    Седьмое частичное произведение
00000000    Восьмое частичное произведение
000000000000000 Перенос
---------------
0000000011001100 Результат (204(10))
```

Прежде всего отметим, что получено восемь частичных произведений, поскольку множитель состоит из 8 разрядов. Первые два частичных произведения включают только нули, так как множители – значения первого и второго разрядов двоичного эквивалента числа 12_{10} – равны 0. Третье частичное произведение – копия множимого. Разница между ними заключается лишь в том,

что копия сдвинута относительно множимого на два двоичных разряда влево, поскольку для получения этого частичного произведения в качестве множителя используется значение третьего разряда. Четвертое частичное произведение также является копией множимого, смещенной относительно последнего на три двоичных разряда влево. Частичное произведение с пятого по восьмое состоит только из нулей, так как соответствующие множители, участвующие в формировании произведения,– двоичные нули. Сложение всех частичных произведений в данном примере не сопровождается переносом, однако возникновение последнего, вообще говоря, не исключено. Результат занимает 16 позиций. Значение восьми старших разрядов равно нулю, поэтому запись полученного произведения можно ограничить восемью младшими значащими разрядами, которых достаточно для представления числа меньшего по величине, чем 255.

Создан простой способ выполнения двоичного умножения, получивший название *умножения путем сдвига и сложения*. Перечислим основные правила этого способа.

1. *Формирование первого частичного произведения*. Если значение младшего значащего разряда множителя равно 0, то и результат равен 0, если значение этого разряда равно 1, то результат является копией множимого.

2. *Правило сдвига*. При использовании очередного разряда множителя для формирования частичного произведения производится сдвиг множимого на один разряд (позицию) влево.

3. *Правило сложения*. Каждый раз, когда значение разряда множителя равно 1, к результату необходимо прибавить множимое, расположенное в позиции, определенной правилом сдвига.

4. *Определение результирующего произведения*. Искомое произведение есть результат выполнения всех операций сдвига и сложения.

Рассмотренный выше пример подтверждает сформулированные правила умножения путем сдвига и сложения. При использовании первого (младшего значащего) разряда множителя множимое не смещается, а поскольку значение этого разряда равно 0, то первое частичное произведение равно 0. Следовательно, равно 0 и текущее значение результата. При использовании второго разряда со значением, равным 0, множимое сдвигается на один разряд влево, но сложение, как и в предыдущем случае, не выполняется. Значение третьего разряда множителя равно 1, поэтому множимое сначала сдвигается еще на один разряд влево, а затем добавляется в качестве слагаемого к текущему значению результата. При использовании четвертого разряда, значение которого равно 1, осуществляется сдвиг множимого на один разряд влево по сравнению с его позицией после операции с третьим разрядом. Затем множимое добавляется к текущему значению результата. Поскольку остальные разряды множителя (с пятого по восьмой) содержат нули, никаких добавлений к текущему значению результата больше не происходит.

Применительно к рассмотренному примеру умножения 17_{10} на 12_{10} правила сдвига и сложения можно продемонстрировать в более компактной форме, а именно:

```
  00010001
  00001100
  --------
    10001      Множимое, сдвинутое влево на
               2 разряда
+
    10001      Множимое, сдвинутое влево на
               3 разряда
  --------
  11001100     Сумма сдвинутых множимых
```

Умножение путем сдвига и сложения существенно упрощает двоичное умножение. Однако это возможно лишь благодаря тому, что при умножении двоичного числа на 0 получается 0, а результат умножения двоичного числа на 1 есть само это число.

Задания для самопроверки

14. Последовательным повторением каких операций можно описать процесс умножения?

15. Положим, что используется традиционный способ умножения чисел. Что формируется при использовании каждого разряда множителя?

16. Выполните умножение приведенных ниже пар двоичных чисел:

а) $\begin{array}{r} 101 \\ \times\ 011 \\ \hline \end{array}$ б) $\begin{array}{r} 0110 \\ \times\ 0111 \\ \hline \end{array}$ в) $\begin{array}{r} 011 \\ \times\ 011 \\ \hline \end{array}$

г) $\begin{array}{r} 0101\,1101 \\ \times\ 0010\,1101 \\ \hline \end{array}$ д) $\begin{array}{r} 0001\,1011 \\ \times\ 1111\,1100 \\ \hline \end{array}$

е) $\begin{array}{r} 0111 \\ \times\ 1000 \\ \hline \end{array}$ ж) $\begin{array}{r} 1001 \\ \times\ 1010 \\ \hline \end{array}$

17. Величина каждого частичного произведения в несколько раз больше предыдущего частичного произведения. Чем определяется это число раз?

18. Чему равно частичное произведение, если значение соответствующего разряда множителя равно 1?

19. Чему равно частичное произведение, если значение соответствующего разряда множителя равно 0?

20. Как называется процедура двоичного умножения?

4.5. Двоичное деление

Деление – это операция, обратная умножению. Иначе говоря, при делении операцию вычитания повторяют до тех пор, пока уменьшаемое не станет меньше вычитаемого. Число этих повторений показывает, сколько раз вычитаемое укладывается в уменьшаемом.

Умножение как последовательность повторяющихся операций сложения и деление как последовательность повторяющихся операций вычитания можно проиллюстрировать на следующих примерах:

Умножение $7 \times 5 = ?$		Деление $35 : 7 = ?$	
0		35	
+		−	
7	1	7	1
7		28	
+		−	
7	2	7	2
14		21	
+		−	
7	3	7	3
21		14	
+		−	
7	4	7	4
28		7	
+		−	
7	5	7	5
35		0	

Пять последовательных операций сложения числа 7 дает в результате число 35, а при выполнении пяти последовательных операций вычитания числа 7 из числа 35 получается 0.

Процедура деления несколько сложнее процедуры умножения. Рассмотрим, например, деление числа 204_{10} на число 12_{10}, пользуясь правилами десятичной и двоичной арифметик:

Десятичное деление		Двоичное деление	
204	12	1100 1100	1100
	|—		|——
12	17	1100	10001
—		——	
84		01	
84		0	
—		——	
0		011	
		0	
		110	
		0	
		——	
		1100	
		1100	
		——	
		0	

Процедура десятичного деления нам знакома и начинается с анализа *делимого* (204) и *делителя* (12). Мы обнаруживаем, что число 12 «укладывается» в числе 20 только один раз, поскольку *остаток* (8) меньше делителя (12). Объединяя остаток со следующей цифрой делимого, получаем число 84 и снова определяем, сколько раз делитель (12) укладывается в 84. Результатом деления является частное, равное 17.

Двоичное деление также начинается с анализа делимого (1100 1100) и делителя (1100). Сразу же обнаруживаем, что делитель (1100) точно укладывается в 1100, а поэтому записываем цифру 1 в поле, предоставленное для формирования частного. Умножаем делитель на 1 и вычитаем

результат из 1100. Разность равна нулю, т. е. меньше делителя, а потому процесс деления можно продолжить. Объединяем нуль остатка со значением следующего разряда делимого, равным 1. Поскольку делитель (1100) укладывается 0 раз в числе 1, записываем 0 в поле представления частного, а число 1 объединяем со следующей цифрой делимого и т. д. Описываемая процедура продолжается до тех пор, пока делимое не оказывается исчерпанным.

Из рассмотренного выше следует, что реализовать операцию деления в вычислительной машине не столь просто, как операцию умножения. Слишком много усилий требуется для выяснения того, сколько раз делитель укладывается в определенном числе. Несмотря на указанные затруднения, был разработан несложный способ двоичного деления, используемый в микропроцессорах.

Процедура двоичного деления в действительности проста, потому что каждый бит частного принимает одно из двух возможных значений: 1 или 0. Как и в случае двоичного умножения, удобным оказывается использование операции сдвига. Продемонстрируем упомянутый способ двоичного деления на рассмотренном выше примере, предварительно представив делитель (12) в дополнительном коде. Это позволит ограничиться двоичным сложением во всех случаях, когда нужно выполнять сложение или вычитание. Дополнительный код двоичного представления числа 12_{10} определяется следующим образом:

```
 01100   Двоичное представление числа 12
 10011   Обратный код двоичного представления числа 12
 00001   Единица, добавляемая к обратному коду
 -----
 10100   Дополнительный код числа 12 (т е число −12)
 ↑  ↑
Бит знака  Биты абсолютной величины
```

Теперь можно приступить к собственно делению. Как и в случае выше рассмотренного так называемого *длинного двоичного деления*, необходимо определить, сколько раз делитель укладывается в числе, образованном соответствующим количеством старших значащих битов делимого. Конечно, микропроцессор не может строить догадок относительно того, сколько раз делитель укладывается в указанном числе. В действительности микропроцессор начинает вычитать делитель из этого числа. Если делитель не будет укладываться в упомянутой части делимого, всегда можно вернуть вычтенные биты обратно делимому. О том, что делитель не укладывается, свидетельствует появление отрицательного результата вычитания (бит знака разности равен 1).

Предпримем попытку выполнить вычитание первый раз:

```
 011001100   Делимое
 101000000   Вычитаемое число 12(10)
 ---------
 000001100   Первый результат
 ↑
Наличие здесь нуля означает, что 1-й бит искомого частного равен 1
                 Частное: 1XXXX
```

Если делитель укладывается в соответствующую часть делимого, бит знака равен 0. Это означает, что результат деления – положительное число. В данном примере это оказалось именно так. А поэтому первый бит искомого результата (частного) равен 1.

Выполним второй шаг процедуры деления. Следует еще раз попытаться выполнить вычитание делителя. Но предварительно необходимо сдвинуть первый результат. Сдвиг должен быть таким, чтобы при последующей операции вычитания был сформирован второй бит частного. В результате сдвига получим

```
 000001100   Первый результат до сдвига
 000011000   Первый результат после сдвига
```

Теперь можно выполнить вторую операцию вычитания:

```
 000011000   Сдвинутый первый результат
 101000000   Вычитаемое число 12(10)
 ---------
 101011000   Второй результат
 ↑
Наличие здесь единицы означает, что 2-й бит искомого частного равен 0
                 Частное: 10XXX
```

Результат этого вычитания содержит 1 в позиции знака, т. е. получено отрицательное число. Следовательно, делитель не укладывается в соответствующем числе. Поэтому прежде всего во вторую (по старшинству) позицию поля представления частного следует записать 0. Кроме того, поскольку вычитание не состоялось, необходимо вернуть биты делителя обратно первому результату:

1 0101100	Второй результат
0 1100000	Возвращаемое число 12_{10}
0 0001100	Сдвинутый первый результат (полученный вторично)

Теперь наступает очередь следующего сдвига:

0 000100	Сдвинутый первый результат
0 001100	Первый результат после второго сдвига

И вновь все готово для попытки уложить делитель в соответствующее число (дважды сдвинутый первый результат). Выполняем третью операцию вычитания:

0 001100	Дважды сдвинутый первый результат
1 010000	Вычитаемое число 12_{10}
1 011100	Третий результат

↑
Наличие здесь единицы означает, что 3-й бит искомого частного равен 0 Частное: 100XX

Третий результат – отрицательное число, т. е. подвергнутый двойному сдвигу первый результат оказался меньше делителя, а следовательно, третий бит искомого частного также равен 0. Поэтому число 12_{10} (делитель) следует вернуть первому результату, сдвинутому дважды:

1 011100	Третий результат
0 110000	Возвращаемое число 12_{10}
0 001100	Дважды сдвинутый первый результат

После исправления допущенной «ошибки» необходимо выполнить очередной сдвиг:

0 001100	Дважды сдвинутый первый результат
0 01100	Первый результат после третьего сдвига

Теперь можно предпринять попытку вычесть делитель из первого результата, подвергнутого троекратному сдвигу:

0 01100	Первый результат после третьего сдвига
1 01000	Вычитаемое число 12_{10}
1 10100	Четвертый результат

↑
Наличие здесь единицы означает, что 4-й бит частного равен 0 Частное: 1000X

Поскольку результат этого вычитания есть число отрицательное, бит частного равен 0, и число 12_{10} необходимо прибавить к четвертому результату:

1 10100	Четвертый результат
0 11000	Возвращаемое число 12
0 01100	Первый результат после третьего сдвига

Выполняем четвертый сдвиг:

0 01100	Первый результат после третьего сдвига
0 1100	Первый результат после четвертого сдвига

Опять предпринимается попытка вычитания делителя из последнего результата. Обратите внимание, что на этот раз число 12_{10} вычитается из числа 12_{10}:

0 1100	Первый результат после четвертого сдвига
1 0100	Вычитаемое число 12_{10}
0 0000	Пятый результат

↑
Наличие здесь нуля означает, что 5-й бит частного равен 1 Частное: 10001

На этом выполнение процедуры заканчивается. Таким образом, при делении числа 1100 1100 на число 1100 результат равен 10001.

Рассмотренная процедура двоичного деления несколько сложнее процедуры

двоичного умножения. Однако ее нетрудно осуществить на практике, если последовательно выполнять предписываемые операции. Поскольку правила деления четко сформулированы, они могут быть реализованы в микропроцессоре. Следует, однако, удостовериться, что эти правила справедливы для всех возможных ситуаций.

Рассмотрим еще один пример, чтобы быть уверенным в правильности понимания описанной выше процедуры. Требуется разделить число 35 на число 5. Процедура так называемого длинного деления двоичных эквивалентов этих чисел имеет следующий вид:

```
100011   101
000      |---
---      0111
1000
 101
-----
00111
  101
  ---
 0101
  101
  ---
  000
```

Представим делитель в дополнительном коде:

```
0 101  Число 5(10)
1 010  Обратный код числа
0 001  Единица, добавляемая к обратному коду
-----
1 101  Дополнительный код числа
```

Теперь опишем процедуру деления методом вычитания и сдвига влево:

Двоичная арифметика	Бит частного	Наименование операндов
0 100011		Делимое 35_{10}
1 011000		Дополнительный код числа 5_{10}
1 111011	0	Первый результат
1 111011		Первый результат
0 101000		Возвращаемое число 5_{10}
0 100011		Делимое
1 00011		Делимое после первого сдвига
1 01100		Дополнительный код числа 5_{10}
0 01111	1	Второй результат
0 0111		Второй результат после сдвига на 1 позицию
1 0110		Дополнительный код числа 5_{10}
0 0101	1	Третий результат
0 101		Третий результат после сдвига на 1 позицию
1 011		Дополнительный код числа 5_{10}
0 000	1	Четвертый результат

Если процедуру деления продолжить и далее, то все последующие биты частного окажутся равными 0. Обычно известно число ожидаемых битов частного и местоположение двоичной точки. Например, при делении 16-битового делимого на 8-битовый делитель частное состоит из 8 бит. Когда известно положение двоичной точки делимого и делителя, то автоматически определяется положение двоичной точки частного.

Задания для самопров[illegible]

21. К повторению каких перечисленных здесь операций сводится деление: а) сложения, б) вычитания, в) умножения или г) всех указанных операций?

22. Объясните, почему для выполнения деления требуется более сложный набор правил, чем в случае умножения?

23. Используя описанную выше процедуру двоичного деления путем вычитания и сдвига, выполните деление следующих положительных чисел: а) 11110/101, б) 10101/111, в) 101000/100, г) 1001/11, д) 1100011/1001, е) 10000100/1100.

Проверьте полученные результаты, используя процедуры двоичного и десятичного длинного деления.

4.6. Арифметика повышенной точности

При работе с микропроцессором часто выясняется, что длина слов, которыми он оперирует, недостаточна для достижения определенной точности вычислений. Это означает, что необходимо представление чисел бо́льших размеров и в такой форме,

чтобы микропроцессор мог работать с ними. Например, широко распространенные 8-разрядные микропроцессоры позволяют использовать числа в диапазоне от -128 до $+127$. Очевидно, что для большинства задач такой диапазон неприемлем. Используя два 8-битовых слова с представлением отрицательных чисел в дополнительном коде, получим диапазон от -32768 до 32767. В этом случае ошибка представления чисел оценивается одной шестидесятитысячной, или $\pm 0{,}0015\%$.

Для решения многих задач указанной *двойной точности* вполне достаточно. Однако иногда требуется еще более высокая точность вычислений, достигаемая, например, тройной точностью представления чисел: 1 бит для знака и 23 бит для абсолютной величины числа.

При использовании в 8-разрядном микропроцессоре тройной точности представления чисел диапазон последних простирается от -8388608 до $+8388607$, включая 0. Это обеспечивает значительно меньшую ишибку представления чисел по сравнению с использованием для этиих целей только 6 бит. Ошибка оценивается величиной меньшей, чем одна миллионная. Однако за все приходится расплачиваться: при работе с арифметикой повышенной точности требуется больший объем памяти для хранения данных и более интенсивная работа микропроцессора. Пусть, например, вы хотите использовать арифметику тройной точности в 8-разрядной вычислительной системе. В этом случае для выполнения сложения недостаточно извлечь из памяти два слова, сформировать их сумму в аккумуляторе и записать результат в 1-байтовую область памяти. Сначала необходимо произвести обращение к младшему значащему байту каждого числа. После сложения двух байтов результат записывают в память, а возможные при этом переносы подлежат временному хранению. Затем из памяти извлекают средние по значимости байты и складывают с битами переноса, полученными в результате предыдущей операции сложения. Результат записывают в память на место, специально зарезервированное для среднего байта суммы. Наконец из памяти извлекают старшие значащие байты, складывают их, к сумме добавляют биты переноса, полученные при предыдущей операции сложения, и результат записывают в область памяти, зарезервированную для старшего значащего байта суммы.

Из рассмотренного примера следует, что сложение тройной точности по сравнению со сложением одинарной точности, т.е. сложением 8-разрядных двоичных чисел, занимает времени в три раза больше; кроме того, при этом требуется в три раза больший объем памяти. И это не единственный недостаток арифметики повышенной точности. Если, например, в процессе сложения чисел тройной точности произойдет прерывание, инициированное другими шагами программы, то необходимо временно сохранить содержимое регистра состояния. В противном случае будут утеряны промежуточные биты переноса, что приведет к неверному результату сложения.

Арифметику повышенной точности можно использовать применительно ко всем четырем основным операциям: сложению, вычитанию, умножению и делению.

Задания для самопроверки

24. Объясните назначение арифметики повышенной точности.

25. Чему равен диапазон чисел двойной точности 16-разрядного микропроцессора при использовании дополнительных кодов?

26. Почему реализация операций арифметики двойной точности сопряжена с дополнительной работой по сравнению с использованием арифметики одинарной точности?

27. Какая точность необходима для представления приводимых ниже чисел в 8-разрядном микропроцессоре?

а) 1568; б) -10264329; в) 22438; г) -129; д) 12348; е) -1000274.

4.7. Арифметика чисел с плавающей точкой

Не все проблемы могут быть разрешены при использовании арифметики повышен-

ной точности. Так, проводимое до настоящего момента рассмотрение было ограничено целыми числами. Мы не знаем, как обращаться с дробной частью числа. Нам неизвестно, как представлять очень большие и очень малые числа.

Перечисленные проблемы разрешимы с помощью *арифметики чисел с плавающей точкой* (*запятой*), позволяющей микропроцессору отслеживать положение десятичной точки. Это достигается благодаря использованию представления десятичных дробей в нормализованном виде, т. е. в виде *мантиссы*, диапазон значений которой простирается от 0,1 до 1, и *порядка* – показателя степени числа 10. Например, число 50 представляется как $0{,}5 \times 10^2$, а число -750 как $-0{,}75 \times 10^3$. Очень малое число 0,00105 записывается в виде $0{,}105 \times 10^{-2}$.

Числа с плавающей точкой хранятся в микропроцессоре так же, как и целые числа. Записывается мантисса со знаком и порядок со знаком. Не следует забывать, что мантисса – число, принадлежащее диапазону 0,1–1, а порядок – показатель степени числа 10.

Представление числа в форме с плавающей точкой в 8-разрядном микропроцессоре можно изобразить схематически следующим образом:

Адрес байта	Содержимое байта	
M + 3 (4-й байт)	+ или −	7-битовый порядок
M + 2 (3-й байт)	+ или −	7 старших битов мантиссы
M + 1 (2-й байт)		8 средних битов мантиссы
M (1-й байт)		8 младших битов мантиссы

Число в форме с плавающей точкой занимает 4 байт. В первом байте расположены 8 младших битов мантиссы, во втором – 8 средних битов, в третьем – 7 старших битов и бит знака. Мантисса представлена как число тройной точности. Четвертый байт занят порядком: 7 бит величины и 1 бит знака. Согласно такому представлению, число с плавающей точкой имеет следующий формат:

$\pm$ XXXXXXXXXXXXXXXXXXXXXXX $\cdot 2^{\pm\,\text{XXXXXXX}}$,

где X – условное обозначение двоичной цифры (бита). Арифметика чисел с плавающей точкой позволяет оперировать числами от $-2^{23} \times N^{127}$ до $+(2^{23}-1) \times N^{127}$, включая такие малые дроби, как $\pm 1 \times N^{-128}$. Значение N обычно равно 2, но иногда и 10. Следовательно, диапазон представления чисел очень большой.

Если микропроцессор «настроен» на работу с числами, представленными в форме с плавающей точкой, то, как правило, все арифметические операции выполняются под управлением *пакета подпрограмм арифметики с плавающей точкой*. Эти подпрограммы обеспечивают размещение числа в 4 байт памяти в соответствии с форматом числа с плавающей точкой. Однако, если число находится в таком «аккумуляторе с плавающей точкой», можно обращаться к подпрограммам выполнения арифметических операций над содержимым этого «аккумулятора» и числами, расположенными в других областях памяти. Большинство пакетов подпрограмм арифметики чисел с плавающей точкой содержат не только подпрограммы выполнения сложения, вычитания, умножения и деления, но и возведения в квадрат, извлечения квадратного корня, вычисления тригонометрических функций (синуса, косинуса, тангенса) и логарифмов. Будучи один раз написанным, пакет таких подпрограмм используется многократно и часто воспринимается пользователями как расширение аппаратных средств микро-ЭВМ. Однако не следует забывать, что, говоря об «аккумуляторе с плавающей точкой», мы имеем в виду четыре смежные области памяти, к которым обращаются подпрограммы арифметики чисел с плавающей точкой.

Пакет подпрограмм арифметики чисел с плавающей точкой выполняется значительно медленнее, чем традиционные команды микропроцессора. Так, время сложения чисел с плавающей точкой в 10–20

раз больше, чем время реализации команды сложения аппаратными средствами микропроцессора. Это объясняется тем, что в сложении чисел с плавающей точкой участвуют от 20 до 30 команд микропроцессора. Подпрограмме приходится не только манипулировать числами тройной точности, но и постоянно следить за значением порядка.

Пакет подпрограмм арифметики чисел с плавающей точкой используется почти всегда, когда микропроцессор управляет работой системы, оперирующей самыми разнообразными числовыми данными. Одним из типичных примеров применения этого пакета является программирование работы микро-ЭВМ на языке БЕЙСИК. Программное обеспечение всех языков высокого уровня, подобных языку БЕЙСИК или ФОРТРАН, выключает свой собственный пакет подпрограмм арифметики чисел с плавающей точкой. Структура таких пакетов весьма сложна по сравнению с их аналогами, используемыми для выполнения ограниченного набора функций системы управления микропроцессора.

Задания для самопроверки

28. Для представления чисел какого размера используется арифметика чисел с плавающей точкой?

29. Представьте в форме мантиссы и порядка следующие числа:

а) 12; б) 222,3; в) $-0{,}334$; г) 1 256 000; д) 0,0000125; е) -1000; ж) $-0{,}000101$; з) 100; и) 22 000 000; к) 0,000000021.

30. Числа в форме с плавающей точкой представляются в машине в виде двух чисел. Назовите их.

Упражнения

4.1. Что происходит при сложении двух цифр, если их сумма не выражается одной цифрой: а) заем, б) суммирование, в) перенос или г) умножение?

4.2. Выполните двоичное сложение следующих ниже слагаемых и укажите, в каких случаях возникает бит переноса:

а) $0+0$; б) $0+1$; в) $1+0$; г) $1+1$.

4.3. При выполнении машиной сложения чисел фиксированной длины возможно появление незначащих нулей. Записываются ли они как а) биты переноса, б) 1, в) биты заема или как г) 0?

4.4. К чему следует прибавлять бит переноса, возникающий в результате выполнения операции сложения?

4.5. Приводимые ниже десятичные числа замените 8- или 16-битовыми двоичными эквивалентами и выполните указанные операции:

а) $\begin{array}{r} 12525 \\ +\ 621 \end{array}$ б) $\begin{array}{r} 2048 \\ +\ 64 \end{array}$ в) $\begin{array}{r} 99 \\ +\ 107 \end{array}$

г) $\begin{array}{r} 1296 \\ +\ 151 \end{array}$ д) $\begin{array}{r} 56274 \\ +\ 32768 \end{array}$ е) $\begin{array}{r} 128 \\ +\ 256 \end{array}$

4.6. Выполните двоичное вычитание и укажите, в каких случаях возникает необходимость заема бита.

а) $0-0$, б) $1-0$, в) $0-1$, г) $1-1$.

4.7. Объясните, откуда следует брать бит заема?

4.8. Выполните двоичные и десятичные операции сложения над следующими парами двоичных чисел:

а) $\begin{array}{r} 010111 \\ -\ 001011 \end{array}$ б) $\begin{array}{r} 0110111 \\ -\ 0001011 \end{array}$ в) $\begin{array}{r} 010101 \\ -\ 000001 \end{array}$

г) $\begin{array}{r} 0001011 \\ -\ 0000111 \end{array}$ д) $\begin{array}{r} 01111111 \\ -\ 01111110 \end{array}$ е) $\begin{array}{r} 0000111 \\ -\ 0000010 \end{array}$

4.9. Какую форму записи вероятнее всего вы используете для представления двоичных положительных и отрицательных чисел:

а) в виде мантиссы и порядка, б) обратный код, в) дополнительный код или г) в виде знака и величины?

4.10. Какая система счисления используется для представления отрицательных чисел в виде обратного кода, дополнительного кода и величины со знаком?

4.11. Каков диапазон значений 16-разрядных двоичных чисел, представляемых в дополнительном коде?

4.12. Каков диапазон значений 12-разрядных двоичных чисел, представляемых в дополнительном коде?

4.13. Представьте указанные ниже десятичные числа их двоичными эквивалентами в дополнительном коде:

а) $+12$; б) $+16$; в) -15; г) $+125$; д) -100; е) $+64$; ж) -70; з) -127; и) 0; к) -1; л) -128; м) $+127$.

4.14. Пользуясь арифметикой дополнительных кодов, выполните операцию вычитания над указанными ниже парами двоичных чисел.

Если разность является отрицательным числом, выразите последнее в дополнительном коде, а также представьте в двоичном коде его абсолютное значение:

а) $\begin{array}{r} 0101 \\ -\ 0111 \\ \hline \end{array}$ б) $\begin{array}{r} 0111\,0111 \\ -\ 0011\,0101 \\ \hline \end{array}$ в) $\begin{array}{r} 0110\,1011 \\ -\ 0111\,1111 \\ \hline \end{array}$

г) $\begin{array}{r} 0101\,1100 \\ -\ 0011\,1011 \\ \hline \end{array}$ д) $\begin{array}{r} 0110\,1101\,1011\,1111 \\ -\ 0111\,1101\,1100\,0101 \\ \hline \end{array}$

е) $\begin{array}{r} 0111\,1111\,1000\,0000 \\ -\ 0000\,1101\,0110\,1100 \\ \hline \end{array}$

4.15. Объясните, в чем заключается основное достоинство процедуры двоичного умножения посредством операций сдвига и сложения.

4.16. Выполните двоичное умножение следующих пар чисел: 0×0, 1×0, 0×1, 1×1.

4.17. Используя операции сдвига и сложения, выполните двоичное умножение следующих пар чисел:

а) $\begin{array}{r} 0000\,1001 \\ \times\ 0000\,0101 \\ \hline \end{array}$ б) $\begin{array}{r} 0100\,0110 \\ \times\ 0001\,0101 \\ \hline \end{array}$ в) $\begin{array}{r} 0101\,0101 \\ \times\ 0101\,0101 \\ \hline \end{array}$

г) $\begin{array}{r} 0101\,1110 \\ \times\ 1000\,0101 \\ \hline \end{array}$ д) $\begin{array}{r} 0111\,0111\,0110\,1110 \\ \times\ 0000\,1010\,0111\,0101 \\ \hline \end{array}$

е) $\begin{array}{r} 1000\,0000\,0001\,0111 \\ \times\ 0010\,0000\,0000\,0111 \\ \hline \end{array}$

4.18. Как принято называть произведение множимого на 1 бит множителя?

4.19. Пользуясь способом, реализованным в микропроцессоре, выполните следующие операции деления:

а) 1100100/01010, б) 10000/0100, в) 9C/D, г) 11110/0010.

4.20. Какой из перечисленных здесь показателей можно улучшить, если использовать арифметику повышенной точности: сложение, выполнение переноса, разрешающую способность или точность?

4.21. Сколько разрядов необходимо для представления абсолютной величины и знака числа с тройной точностью в 8-разрядном микропроцессоре?

4.22. Известно, что представление чисел в форме с плавающей точкой вызывает использование арифметики повышенной точности. Как при этом выглядит запись такого числа?

4.23. Ряд операций арифметики чисел в форме с плавающей точкой выполняется пакетом подпрограмм этой арифметики. Какие из следующих операций: сложение, вычисление синуса, логарифмирование или установка начальных значений – не выполняются этим пакетом?

Ответы на вопросы заданий для самопроверки

1. а) 0001 0110; б) 1001 1000; в) 1111 1101; г) 1111 1111; д) 1000; е) 1100; ж) 1010 0001 000 0010; з) 1001 0111 1011 1000.

2. а) $\begin{array}{r} 0110\,0101 \\ +\ 0001\,0000 \\ \hline 0111\,0101 \end{array}$ б) $\begin{array}{r} 1110\,0001 \\ +\ 1010\,1000 \\ \hline 1\,1000\,1001 \end{array}$

в) $\begin{array}{r} 0000\,0001\,1000\,1110 \\ +\ 0000\,0000\,1000\,0100 \\ \hline 0000\,0010\,0001\,0010 \end{array}$ г) $\begin{array}{r} 0011\,1000 \\ +\ 0000\,1010 \\ \hline 0100\,0010 \end{array}$

д) $\begin{array}{r} 0101\,0110 \\ +\ 0001\,1001 \\ \hline 0110\,1111 \end{array}$ е) $\begin{array}{r} 0000\,0001\,0010\,0001 \\ +\ 0000\,0001\,1110\,1101 \\ \hline 0000\,0011\,0000\,1110 \end{array}$

3. 9 разрядов.

4. Числа 0101 и 0000 0101 отличаются только незначащими нулями, что не оказывает никакого влияния на результат суммирования.

5. а) 0100; б) 0010; в) 0110; г) 1000; д) 0010 1101; е) 1110 1110; ж) 0011 0101 1010 1001.

6. а) $\begin{array}{r} 1111 \\ -\ 1100 \\ \hline 0011 \end{array}$ б) $\begin{array}{r} 1000 \\ -\ 0010 \\ \hline 0110 \end{array}$ в) $\begin{array}{r} 1111\,1111 \\ -\ 0001\,1000 \\ \hline 1110\,0111 \end{array}$

г) $\begin{array}{r} 0011\,0100 \\ -\ 0010\,0100 \\ \hline 0001\,0000 \end{array}$ д) $\begin{array}{r} 0001\,1101 \\ -\ 0000\,1100 \\ \hline 0001\,0001 \end{array}$ е) $\begin{array}{r} 1000\,1000 \\ -\ 0110\,1100 \\ \hline 0001\,1100 \end{array}$

7. Заем (отрицательный перенос) возникает при попытке вычитания большей цифры из меньшей.

8. Заем возникает при вычитании, а перенос при сложении, и называются они арифметическим переполнением.

9. б.

10. а) 0100 1001; б) 1001 0100; в) 010; г) 1111 0011; д) 0; е) 1111 1110; ж) 0000 0000; з) 1010 1010; и) 0000 1111 0000 1111; к) 0011 0011 0011 1100.

11. а) Отрицательное, б) положительное, в) положительное, г) положительное, д) положительное, е) положительное, ж) отрицательное, з) положительное, и) отрицательное, к) отрицательное.

12. а) 0100 0000; б) 1100 1000; в) 0000 1100; г) 0000 0000; д) 1000 0000; е) 0111 1111; ж) 0010 0000; з) 1110 0000; и) число 256 нельзя

представить в виде 8-разрядного двоичного числа; если же использовать 16 разрядов, то оно принимает следующий вид: 0000 0001 0000 0000; к) 0001 0000; л) 1001 1100; м) 1111 1100.

13. В обоих случаях применяются одни и те же правила решения, поскольку указанные двоичные представления эквивалентны. Например, 1100 можно рассматривать как число 12 или как дополнение числа 4:

$$\begin{array}{r} 0111 \\ +\,1100 \\ \hline 10011 \end{array} \quad \begin{array}{r} 7 \\ +\,12 \\ \hline 19 \end{array} \quad \text{или} \quad \begin{array}{r} 0111 \\ +\,1100 \\ \hline 10011 \end{array} \quad \begin{array}{r} 7 \\ -\,4 \\ \hline 3 \end{array}$$

Во втором случае бит переноса (крайний слева) игнорируется, так как используется арифметика дополнительных кодов.

14. Последовательным повторением операций сложения.

15. Частичное произведение.

16.

а)
$$\begin{array}{r} 101 \\ \times\,011 \\ \hline 101 \\ 101 \\ 000 \\ \hline 01111 \end{array}$$

б)
$$\begin{array}{r} 0110 \\ \times\,0111 \\ \hline 0110 \\ 0110 \\ 0110 \\ 0000 \\ \hline 0101010 \end{array}$$

в)
$$\begin{array}{r} 011 \\ \times\,011 \\ \hline 011 \\ 011 \\ 000 \\ \hline 01001 \end{array}$$

г)
$$\begin{array}{r} 01011101 \\ \times\,00101101 \\ \hline 01011101 \\ 00000000 \\ 01011101 \\ 01011101 \\ 00000000 \\ 01011101 \\ 00000000 \\ 00000000 \\ \hline 0001000001011001 \end{array}$$

д)
$$\begin{array}{r} 00011011 \\ \times\,11111100 \\ \hline 00000000 \\ 00000000 \\ 00011011 \\ 00011011 \\ 00011011 \\ 00011011 \\ 00011011 \\ 00011011 \\ \hline 0001101010010100 \end{array}$$

е)
$$\begin{array}{r} 0111 \\ 1000 \\ \hline 0000 \\ 0000 \\ 0000 \\ 0111 \\ \hline 0111000 \end{array}$$

ж)
$$\begin{array}{r} 1001 \\ 1010 \\ \hline 0000 \\ 1001 \\ 0000 \\ 1001 \\ \hline 1011010 \end{array}$$

17. В n раз, где n – основание системы счисления.

18. Равно множимому.

19. Равно нулю.

20. Называется процедурой путем сдвига и сложения.

21. б.

22. Деление сложнее умножения, так как в первом случае путем сопоставления делимого и делителя необходимо подбирать значение частного, а затем проверять корректность сделанного выбора. Такой анализ превышает возможности процессора.

23. а) 110; б) 11; в) 1010; г) 11; д) 1011; е) 1011.

24. Повышенная точность представления чисел используется в тех случаях, когда одного слова недостаточно для записи числа.

25. $\pm 2^{31} = \pm 2\,147\,483\,648$.

26. При выполнении операций арифметики двойной точности одновременно обрабатывается только одно слово, и, следовательно, необходимо временно хранить много промежуточных данных и следить за всеми переносами и заемами.

27. а) Двойная, б) четырехкратная, в) двойная, г) двойная, д) двойная, е) тройная.

28. Большие или малые числа.

29. а) 0.12×10^2; б) 0.2223×10^3; в) -0.334×10^0; г) 0.1256×10^7; д) 0.125×10^{-4}; е) -0.1×10^4; ж) -0.101×10^{-3}; з) 0.1×10^3; и) 0.22×10^8; к) 0.21×10^{-7}.

30. Мантисса и порядок.

Глава 5.
Введение в программирование

В настоящей главе излагаются основы программирования микропроцессоров. Показано, что оно имеет много общего с программированием микро-ЭВМ, мини-ЭВМ и больших вычислительных машин. Во всех случаях выбор последовательности операций, обеспечивающих решение поставленной задачи, осуществляет программист.

Вводится понятие программного обеспечения как совокупности программ и документации, используемых процессором для решения прикладных задач. Обычно программное обеспечение хранится в памяти вычислительной системы в виде данных; для пользователя оно представляется в наглядной форме как тексты программ, отпечатанные на бумаге.

В первом разделе читатель знакомится с общим описанием алгоритма решения задачи и пошаговым программированием этого решения, состоящего из арифметических или логических операций. Во втором разделе описывается техника составления блок-схем алгоритмов, которые являются исходным документом для написания текста программы. Блок-схемы алгоритмов – это графическое описание работы программного обеспечения вычислительной системы. В третьем разделе излагаются принципы модульного программирования; здесь показано, что использование подпрограмм делает главную программу более компактной. Четвертый, заключительный раздел главы содержит краткий обзор языков программирования четырех типов: машинного, ассемблера, языков высокого уровня, ориентированных на интерпретацию и компилирование. Объясняется назначение языков каждого типа.

5.1. Что такое программирование?

Часто программирование воспринимают как некую «черную магию», хотя для этого нет совершенно никаких оснований. Говоря простым языком, программирование – это описание последовательности действий, которые вычислительная машина должна выполнить для решения поставленной задачи. А для этого программист должен «говорить» с машиной на понятном ей языке. Знакомство с предыдущими главами показывает, что микропроцессор точно выполняет каждую команду, полученную от программиста. При этом важно подчеркнуть, что микропроцессор выполняет только то, что ему предписывает программист, и не более. Вот почему список команд должен быть исключительно точным. Программист должен сообщить микропроцессору абсолютно все, что следует делать, и точно, шаг за шагом, описать, как это выполнять.

Имеется одно очень важное обстоятельство, которое программист должен ясно себе представлять: если не известно, как решать задачу, то программирование последней для решения на ЭВМ бессмыслен-

но – машина не сможет ее решить. Это не означает, что программист должен знать ответ на поставленную задачу, однако он должен понимать, как его получить. Например, если задача сводится к отысканию корней уравнения, то для получения ответа необходимо ввести в машину численные значения всех переменных. Однако вычислительная машина не может осуществить соответствующие математические выкладки для вывода уравнения, решение которого и есть искомый ответ.

Для программирования работы вычислительной машины требуется описание процесса решения задачи с учетом возможностей машины. В некоторых случаях процедура программирования проста и сводится к записи словесного описания решения задачи на языке алгебраических формул. В других случаях программист располагает только информацией о входных данных и желаемом результате, а от него требуется определение последовательности операций над этими данными с целью получения требуемого результата.

Описание процесса решения задачи таким образом, чтобы ее могла решить вычислительная машина, является основной частью того, что называют *программированием*. Результат такого описания решения задачи именуют *алгоритмом*. Если алгоритм составлен, то его можно закодировать, т. е. представить в виде последовательности машинных команд, осуществляющих решение задачи. Поскольку машина определенной модели имеет свой специфический набор команд, программист должен писать программу, пользуясь только этими командами.

Следующей проблемой, возникающей перед программистом, является *отладка* программы. Для этого программу необходимо загрузить в машину и попытаться ее выполнить. Ошибки, выявляемые в процессе отладочного прогона программы, подлежат исправлению. Для упрощения процедуры отладки программу часто разделяют на небольшие части, каждая из которых подлежит индивидуальной отладке. По мере завершения отладки отдельных частей программы их объединяют, и этот процесс продолжается до полного окончания отладки всей программы.

Составить программу означает описать алгоритм на языке вычислительной машины, т. е. посредством команд, которые машина способна выполнять. Каждая ЭВМ или микро-ЭВМ располагает своим набором команд. Запись алгоритма посредством команд называют *кодированием*, а используемые для этого команды – *исходным кодом*. Поскольку для функционирования машины нужны двоичные команды, исходный код подлежит трансляции (переводу) в так называемый *объектный код*, являющийся двоичным представлением исходного кода.

Рассмотрим пример. Прежде всего сформулируем задачу, подлежащую решению. Требуется построить семейство точек в системе координат X и Y при условии, что для каждой точки отсчет по оси Y равен возведенному в квадрат отсчету по оси X. Значения переменной X – это целые числа, принадлежащие диапазону от -3 до $+3$.

Согласно формулировке задачи для переменной X, принимающей значения -3, -2, -1, 0, 1, 2 и 3, требуется решать уравнение $Y = X^2$.

Словесное описание алгоритма решения можно представить следующим образом:

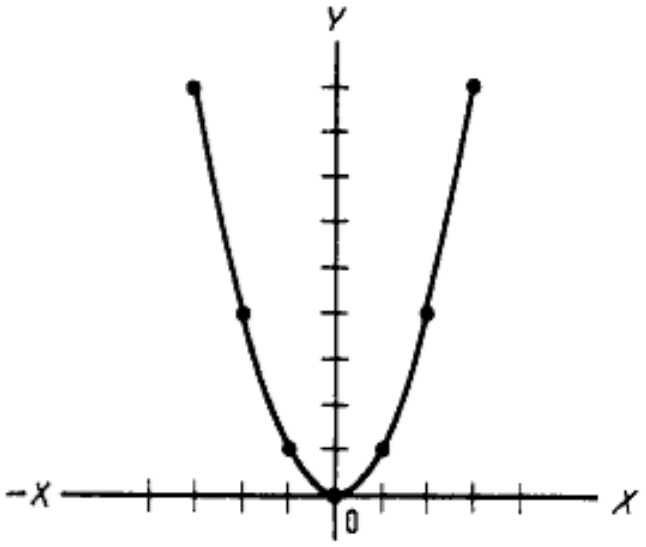
1. Положить $X = -3$
2. Вычислить $Y = X^2$
3. Вывести на печать X, Y
4. Положить $X = -2$
5. Вычислить $Y = X^2$
6. Вывести на печать X, Y
7. Положить $X = -1$
8. Вычислить $Y = X^2$
9. Вывести на печать X, Y
10. Положить $X = 0$
11. Вычислить $Y = X^2$
12. Вывести на печать X, Y
13. Положить $X = 1$
14. Вычислить $Y = X^2$
15. Вывести на печать X, Y
16. Положить $X = 2$
17. Вычислить $Y = X^2$
18. Вывести на печать X, Y
19. Положить $X = 3$
20. Вычислить $Y = X^2$
21. Вывести на печать X, Y
22. Конец

Соответствующая этому алгоритму программа работы вычислительной машины чрезвычайно проста. На рис. 5.1, *а* показана таблица выводимых на печать результатов, на рис. 5.1, *б* – график зависимости Y от X, построенный программистом. Кривая по форме представляет собой параболу.

Анализируя этапы решения рассмотренной задачи, можно сделать следующие выводы: во-первых, на основании словесной

X	Y
−3	9
−2	4
−1	1
0	0
+1	1
+2	4
+3	9

а



б

Рис. 5.1. Результаты работы программы вычисления $Y = X^2$, представленные в табличной и графической формах.

постановки задачи сформировано уравнение; во-вторых, в соответствии с полученным уравнением составлено словесное описание алгоритма решения задачи в виде последовательности команд для вычислительной машины; в-третьих, путем выполнения операций, предписываемых этими командами, получен требуемый результат в виде таблицы выведенных на печать данных и построенного вручную графика.

Рассмотрим еще один пример. Требуется «очистить» первые четыре тысячи областей, т.е. присвоить нулевое значение каждому биту этих областей. Для решения этой задачи необходимо использовать возможности, которые предоставляет микропроцессор для манипулирования данными: загружать в смежные области памяти слова, составленные из двоичных нулей; в качестве «источника» этих слов используем аккумулятор микропроцессора. Словесное описание алгоритма решения этой задачи выглядит следующим образом:

1. Присвоить нулевое значение содержимому всех разрядов аккумулятора.
2. Установить содержимое регистра адреса памяти равным адресу первой области памяти.
3. Записать копию содержимого аккумулятора в область памяти, адресуемую регистром адреса памяти.
4. Установить содержимое регистра адреса памяти равным адресу следующей области памяти.
5. Проверить, равно ли содержимое регистра адреса памяти числу 4001.
6. Если равно, то операции прекратить; в противном случае перейти к выполнению п. 3.

Из описания алгоритма следует, что операции продолжаются до тех пор, пока не будут записаны нули во все четыре тысячи областей памяти. Как только содержимое регистра адреса памяти становится равным 4001, выполнение программы заканчивается. В данном примере программа работы вычислительной машины содержит команды выполнения логических операций, в то время как в предыдущем примере использованы только арифметические операции. При этом в обоих примерах, исходя из постановки задачи, создается словесное описание алгоритма, кодируемое на языке программирования в виде последовательности команд, которые может выполнять машина.

Задания для самопроверки

1. Составьте алгоритм решения с помощью ЭВМ формулируемой ниже задачи.

В лаборатории стандартов необходимо поддерживать температуру равной 20°С независимо от температуры снаружи помещения. В вашем распоряжении имеются кондиционер, отопительная система и два термостата: один – для измерения темпера-

туры, равной или меньше 20°С, другой – для измерения температуры больше или равной 20°С.

2. Пользуясь шагами, подобными приведенным в данном разделе для решения первой задачи, составьте алгоритм решения формулируемой ниже задачи.

Для целочисленной переменной X, принимающей все значения в диапазоне от 2 до 5, постройте графики функций X^2, X^3, $\sqrt{X}$ и $\sqrt[3]{X}$.

3. Что принято относить к программному обеспечению микропроцессора: а) постоянную память, б) магнитную ленту, в) подпрограммы или г) перфоленту?

4. Какова основная цель программирования: а) формирование последовательности команд на языке машины; б) составление уравнения, выражающего решение задачи в математических терминах; в) получение соответствующего аппаратного эквивалента или г) построение графиков зависимости результатов от исходных данных?

5. Какие из нижеперечисленных действий необходимо выполнить после того, как составлен алгоритм решения задачи: а) создать программное обеспечение ЭВМ, б) записать алгоритм на языке программирования, в) проверить корректность постановки задачи или г) прогнать программу на ЭВМ?

6. Объясните, почему вычислительная машина не может решить задачу, если вы не представляете, как можно ее решить?

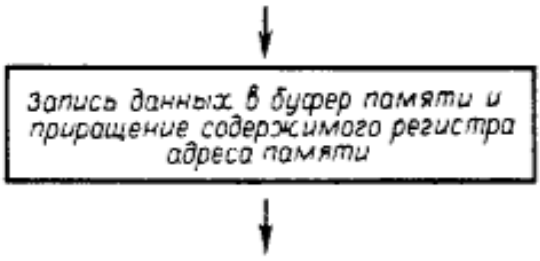
5.2. Составление блок-схем алгоритмов

Графическое изображение алгоритма решения задачи в виде блок-схемы - важный этап подготовки задачи к решению на ЭВМ. Блок-схема алгоритма помогает пользователю адекватно представить работу программы. Анализируя блок-схему, можно выяснить, как различные входные данные влияют на окончательный результат.

Блок-схема алгоритма - одна из важных частей документации, подготавливаемой для решения задачи на ЭВМ. Однако алгоритмизация и графические средства изображения алгоритмов не являются прерогативой вычислительной техники; они могут использоваться для описания решения любых задач. Например, вы можете составить блок-схему алгоритма вашего поведения утром – от момента подъема с постели до ухода на работу или в учебное заведение.

Блок-схема алгоритма составляется из отдельных блоков. Различают четыре типа блоков, каждый из которых имеет один или несколько входов и один или несколько выходов (рис. 5.2)[1]. Стрелками обозначают направление хода вычислений. Блок в форме прямоугольника символизирует выполнение каких-либо операций по обработке данных; текст внутри блока является кратким описанием этого процесса обработки (рис. 5.2, *а*). Например, если в блок-схеме алгоритма содержится блок

то это означает, что на данном этапе работы машины аккумулятор должен быть очищен. В таком случае для выполнения этой операции микропроцессору требуется одна команда. Однако если блок-схема алгоритма включает блок



то для выполнения этих операций микропроцессор использует несколько команд.

Рассмотренные примеры блоков обработки данных демонстрируют возможность описания как простых операций, выполняемых одной командой микропро-

[1] Отметим, что только блок проверки условия, изображаемый в виде ромба, может иметь несколько выходов.– *Прим. перев.*

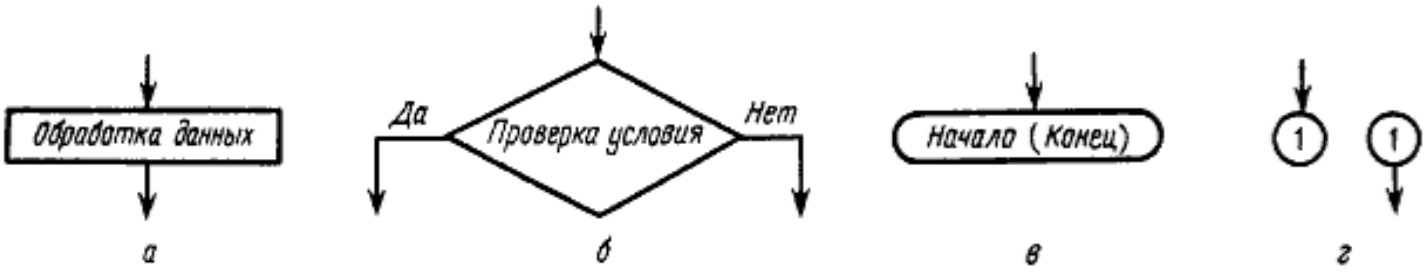
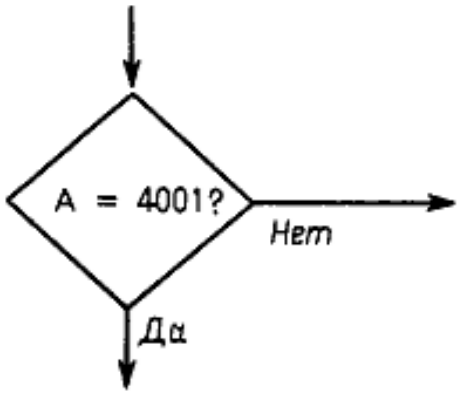


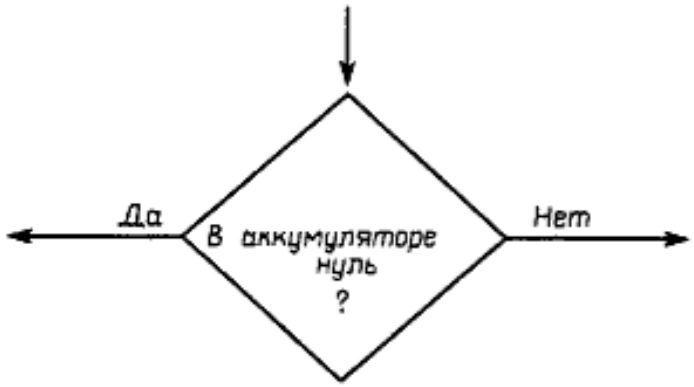
Рис. 5.2. Типы блоков, используемых для составления блок-схем алгоритмов: *а*) блок обработки данных; *б*) блок проверки условия; *в*) блок начала или конца алгоритма; *г*) соединители.

цессора, так и сложных операций, для реализации которых требуется целый набор команд.

Блок, имеющий форму ромба (рис. 5.2, *б*), используется для символического обозначения проверки выполнения какого-либо условия с целью принятия решения о направлении последующего хода вычислений. Внутри блока описывается условие, подлежащее проверке в той точке схемы алгоритма, где размещается данный блок. Возможные результаты проверки указываются на выходах – линиях, выходящих из блока. Например, блок-схема алгоритма содержит блок



Возможны только два результата проверки: переменная A либо равна 4001, либо не равна. В первом случае выполнение алгоритма продолжается в направлении, указываемом стрелкой с надписью «Да», во втором случае – по стрелке с надписью «Нет». Для проведения подобной проверки требуется использование нескольких команд микропроцессора. Однако возможны и более простые операции проверки того или иного условия, реализуемые одной командой. Например, если блок-схема алгоритма включает блок



то достаточно использовать одну команду проверки значения бита нулевого результата в регистре состояния микропроцессора. Если этот бит равен 1, то вычисления продолжаются в направлении линии с надписью «Да»; если же этот бит равен 0, то в направлении линии с надписью «Нет».

Блок овальной формы используется для символического обозначения начала или конца алгоритма (рис. 5.2, *в*). Текст внутри блока, как правило, состоит из одного слова – «Начало» или «Конец».

В тех случаях, когда необходимо «разорвать» линию потока вычислений, идущую от одного блока к другому, применяются так называемые соединители в виде окружности с указанной внутри нее цифрой или буквой (рис. 5.2, *г*). Наличие другого идентичного соединителя (с той же цифрой или буквой) означает, что прерванная в месте расположения первого соединителя линия продолжается с того места, где находится второй подобный соединитель. Использование соединителей упрощает внешний вид блок-схемы алгоритма, что позволяет избежать пересечения линий и дает возможность размещать блок-схему алгоритма на нескольких страницах и т. п.

Рассмотрим теперь в качестве примера блок-схему алгоритма вывода на печать таблицы значений X и Y, связанных между

собой функциональной зависимостью в виде параболы (см. пример, приведенный в разд. 5.1). Блок-схема алгоритма представлена на рис. 5.3. Она состоит из простой последовательности блоков, линия соединения между которыми в одном месте разорвана для удобства размещения схемы на бумаге, а нарушенная связь восстановлена посредством соединителей. Данная схема является примером графического изображения так называемых *прямолинейных алгоритмов*, описывающих простую последовательность действий без ответвлений.

Более сложные алгоритмы содержат

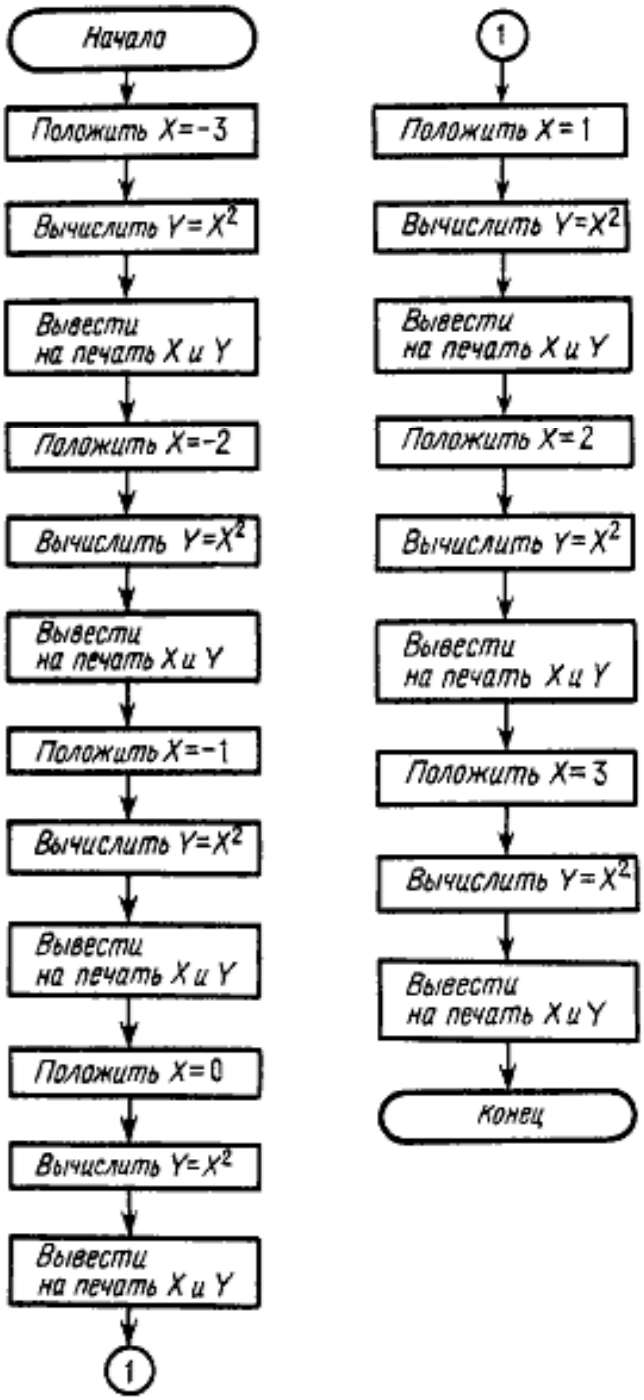


Рис. 5.3. Блок-схема простого алгоритма в виде цепочки следующих друг за другом блоков.

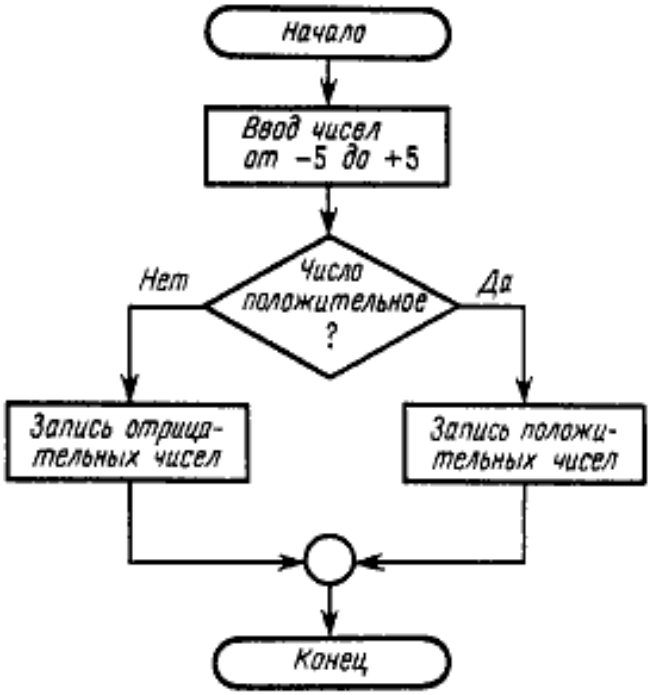


Рис. 5.4. Блок-схема алгоритма процесса. (Соединитель включен в схему для лучшей графической наглядности процесса; его использование необязательно.)

блоки проверки некоторых условий и соответствующие разветвления хода вычислений, а также повторяющиеся циклы команд. В блок-схемах таких алгоритмов используются блоки проверки условия с целью принятия одного из возможных решений. Примером таких алгоритмов является задача формирования двух наборов целых чисел, лежащих в диапазоне от -5 до $+5$. Путем сортировки организуют два набора, один из которых включает положительные числа, а другой – отрицательные. Блок-схема алгоритма решения этой задачи показана на рис. 5.4. Здесь соединитель используется для сведения двух ветвей схемы к одному блоку – символическому концу алгоритма.

Если процесс решения задачи носит итерационный характер, то в соответствующей блок-схеме алгоритма должно иметь место многократное повторение одинаковых действий с проверкой каждый раз одних и тех же условий. Таким является, например, алгоритм очистки смежных областей памяти, описанный в разд. 5.1. Блок-схема этого алгоритма показана на рис. 5.5. Выполняя шаг за шагом действия, предписываемые блок-схемой, можно хорошо понять, что такое *итерационный процесс*. В начале размещается блок присвое-

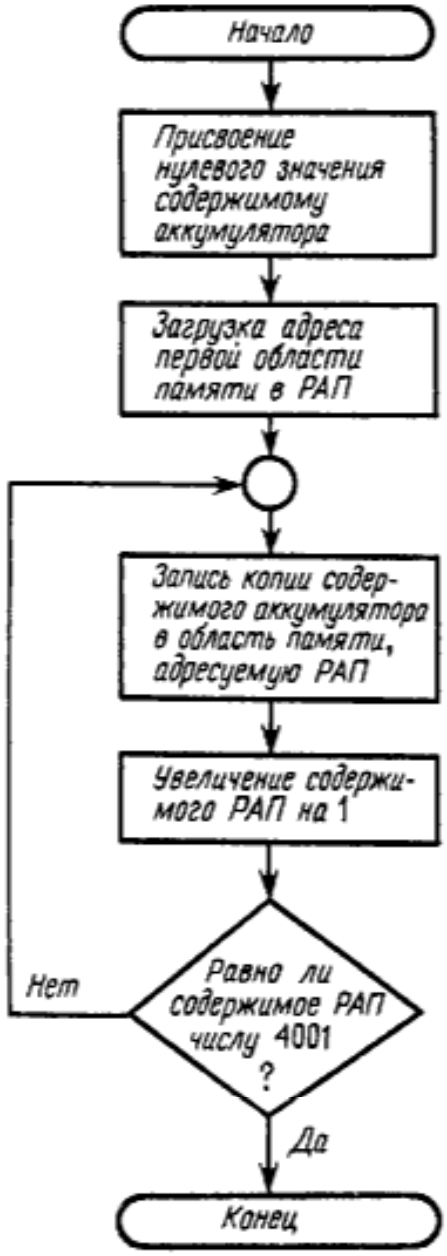


Рис. 5.5. Блок-схема алгоритма итерационного процесса. (Блоки, охваченные петлей обратной связи, включают операции, повторяемые 4002 раза.)

ния нулевых значений содержимому всех разрядов аккумулятора. Затем следует блок загрузки адреса первой области памяти в регистр адреса памяти (РАП). От этого блока через соединитель осуществляется переход к блоку записи копии содержимого аккумулятора в область памяти, адресуемую регистром адреса памяти. После этого выполняется переход к блоку увеличения на единицу содержимого РАП с целью последующей адресации следующей области памяти. Теперь возникает необходимость проверить, не стало ли содержимое РАП равным 4001. Это выполняется после перехода к блоку, изображенному на схеме в форме ромба, т. е. к блоку проверки условия. Если результат указанной проверки является положительным, то осуществляется переход к блоку прекращения операций (блоку останова). Если результат проверки отрицательный, то линия хода операций в форме петли обратной связи замыкается через соединитель (согласно стрелкам) на блок записи копии содержимого аккумулятора в область памяти, адресуемую регистром адреса памяти. Конечно, в этом случае указанный адрес на 1 больше предыдущего. И опять содержимое регистра адреса памяти увеличивается на 1, а затем подлежит проверке. Действия, охватываемые петлей обратной связи, повторяются 4002 раза, после чего все операции прекращаются.

Блок-схема алгоритма на рис. 5.5 содержит все типичные элементы алгоритмов, рассмотренных ранее: цепочки последовательно выполняемых операций, проверку некоторого условия, разветвление хода выполнения операций как следствие принятия решения по результатам проверки. Отличительной особенностью данного алгоритма является наличие петли обратной связи. Упомянутым выше элементам алгоритма на блок-схеме последнего соответствуют блоки обработки, проверки условия, начала и конца алгоритма, а также соединители межблочных линий.

Анализируя рассмотренные блок-схемы алгоритмов, следует отметить, что графическое изображение алгоритма является простым и наглядным способом документирования процесса описания решения задачи с помощью ЭВМ. В дальнейшем вы убедитесь в том, что алгоритм решения любой задачи можно описать, пользуясь такими понятиями, как последовательность операций, проверка условия с последующим принятием решения о направлении хода вычислений и итерационный процесс. Безусловно, блок-схемы алгоритмов могут быть значительно сложнее рассмотренных выше и содержать блоки, отличающиеся по форме от изображенных на рис. 5.2. Иногда для символического изображения специфических процессов вводят особые обозначения. Например, для представления процедур ввода-вывода используют блок в форме параллелограмма. Некоторые программисты применяют специальную символику, поясняющую блок-

схему алгоритма. Если введение дополнительных обозначений позволяет достичь лучшего отображения алгоритма решения задачи, отступление от принятых обозначений вполне допустимо.

Задания для самопроверки

7. Изобразите в виде блок-схем алгоритмов решения задач пп. 1 и 2 задания для самопроверки. Не пытайтесь точно представить на схеме команды машины, но обязательно опишите все, что необходимо для получения требуемых результатов.

8. Какие элементы алгоритмов присутствуют в блок-схемах, разработанных согласно п. 7: последовательности операций, проверки условия, итерационные процессы в виде петли обратной связи? Объясните целесообразность выбранных вами элементов.

9. Составьте блок-схему алгоритма решения следующей задачи: в ЭВМ вводятся два положительных числа – A и B; если A больше B, производится установка в 1 флажка A, в противном случае – флажка B, после чего действия прекращаются.

10. На рис. 5.6 изображена блок-схема алгоритма части некоторой программы, о чем свидетельствуют слова «Вход» и «Выход» внутри символов, обозначающих начало или конец алгоритма и содержащих обычно записи «Начало» или «Конец». В данной схеме линия, выходящая из блока увеличения содержимого регистра B на 1, подходит к соединителю, через который она поступает на вход блока проверки условия, а не блока присвоения содержимому регистра B значения, равного 1000. Какой элемент на схеме свидетельствует именно о такой последовательности действий?

11. Алгоритм, блок-схема которого изображена на рис. 5.6, описывает операции, выполняемые так называемой *подпрограммой задержки*. Дайте словесное описание операций, выполняемых этой программой, и поясните, почему допустимо использование термина «задержка».

12. Изобразите блоки графического представления алгоритма, соответствующие следующим операциям: а) проверка, является ли X больше Y; б) очистка регистра C; в) извлечение квадратного корня из X; г) 100 мс?; д) начало; е) соединение с использованием буквы G; ж) останов; з) ввод A?; и) ввод A; к) выход; л) вычисление выражения $Y = AX^2 + BX + C$.

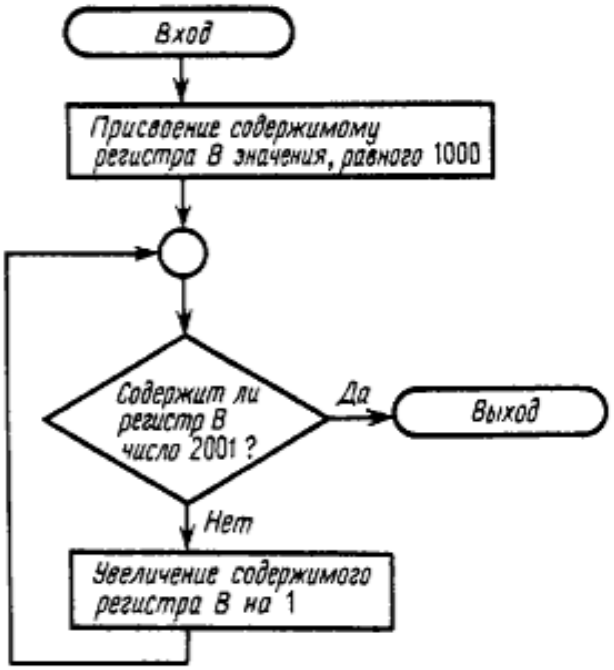


Рис. 5.6. Блок-схема алгоритма к ответу на вопрос п. 10 заданий для самопроверки.

5.3. Подпрограммы

Подпрограмма – это часть программы, используемая обычно несколько раз в процессе выполнения программы. Однако текст подпрограммы записывается программистом только один раз. Когда же программисту необходимо воспользоваться подпрограммой, достаточно указать в программе соответствующую *команду вызова* (обращения к подпрограмме), адресующуюся к области памяти, в которой расположена подпрограмма. Большинство языков программирования располагает для этих целей специальными командами вызова подпрограмм CALL, которые не только инициируют выполнение подпрограмм, но и побуждают вычислительную машину запоминать состояние программы в момент обращения к подпрограмме. В результате вызова подпрограммы ей передается управление с целью пошагового выполнения ее операций. Последней выполняемой командой подпрограммы, как

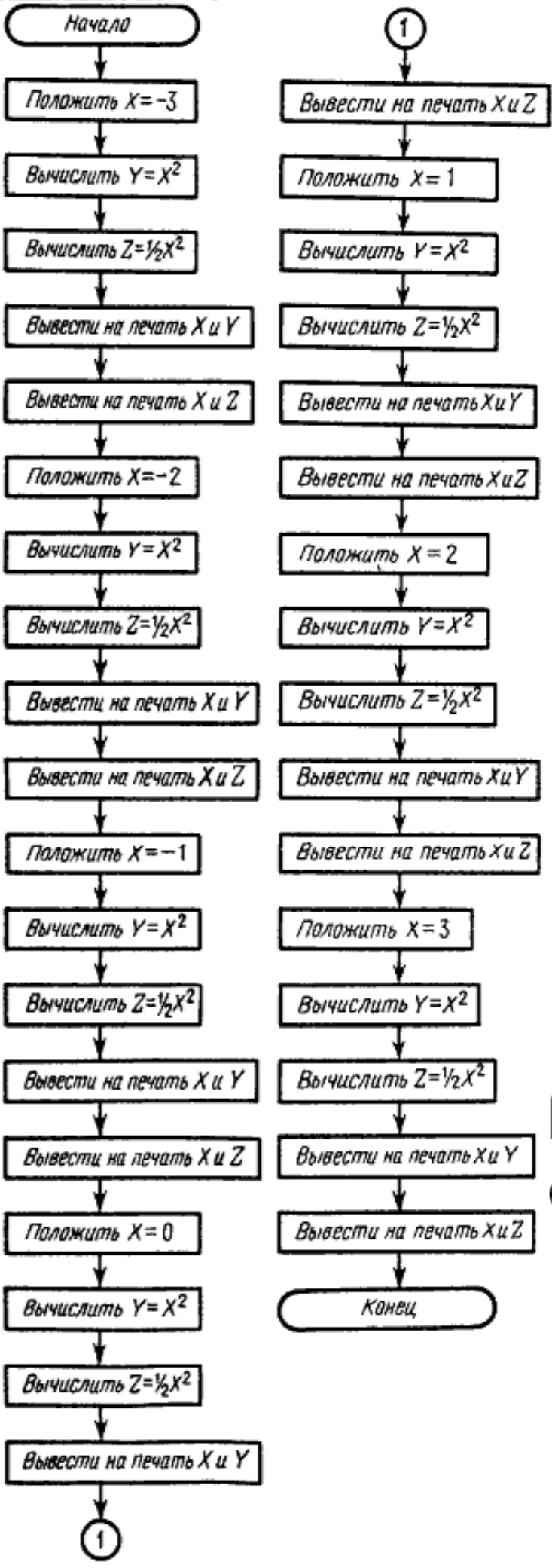


Рис. 5.7. Блок-схема алгоритма последовательного вычисления и вывода на печать значений аргументов и функций $Y = X^2$ и $Z = {}^1/_2X^2$.

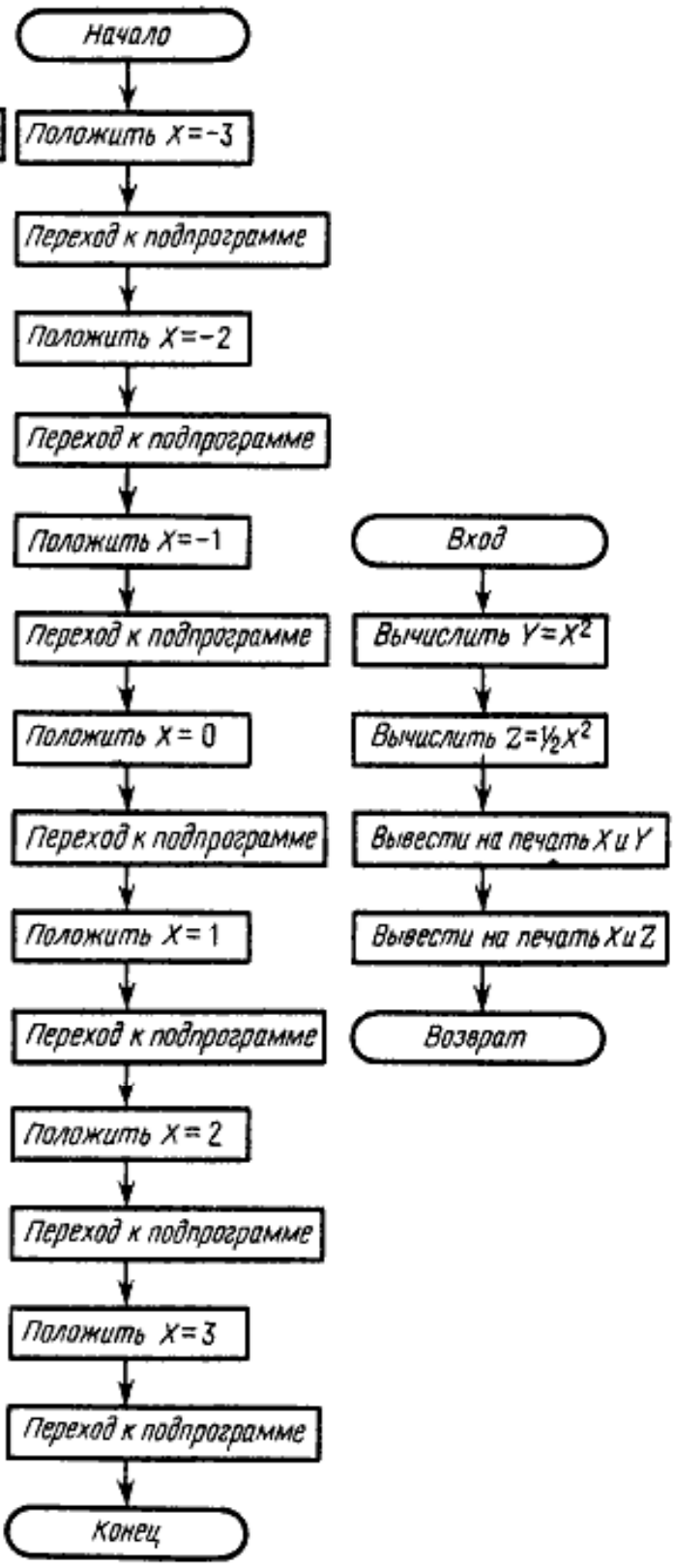


Рис. 5.8. Блок-схема алгоритма вычисления и вывода на печать значений аргументов и функций $Y = X^2$ и $Z = {}^1/_2X^2$ с использованием подпрограммы.

правило, является так называемая *команда возврата* RETURN, передающая управление обратно вызвавшей программе, заставляя последнюю продолжить выполнение с той команды, которая непосредственно следует за командой вызова подпрограммы.

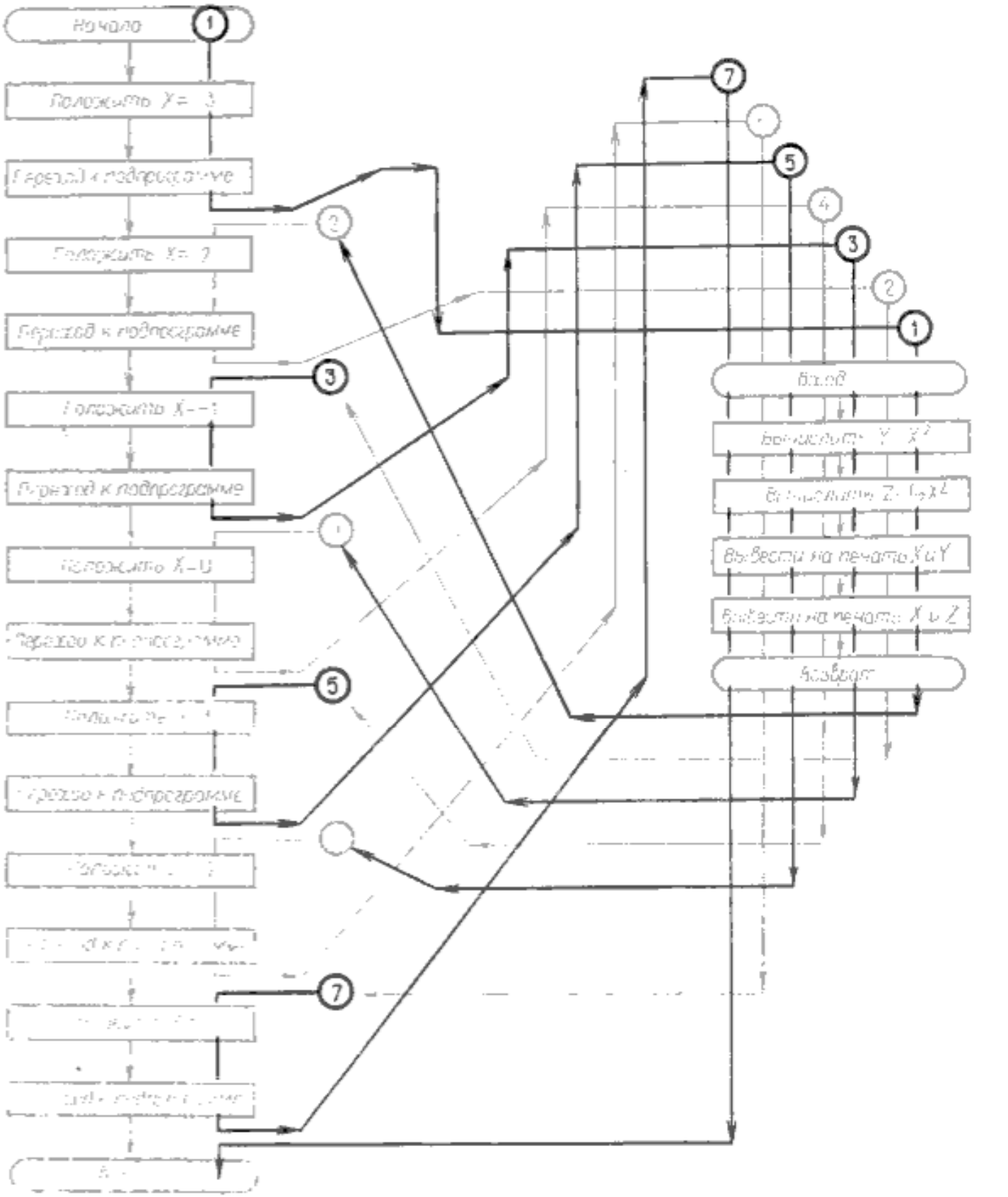


Рис. 5.9. Схематическое изображение семикратного обращения к подпрограмме в процессе выполнения главной программы.

Основное достоинство подпрограмм заключается в том, что благодаря возможности их многократного использования сокращается текст программы в целом. Вместо того чтобы по мере необходимости повторять запись одного и того же фрагмента программы, достаточно оформить запись фрагмента как подпрограмму и обращаться к ней столько раз, сколько требуется в соответствии с алгоритмом решения задачи.

Рассмотрим пример использования под-

программы. Обратимся еще раз к задаче формирования значений функции, имеющей форму параболы при построении графика в декартовой системе координат. Расширим постановку задачи требованием вычисления значений двух парабол: $Y = X^2$ и $Z = {}^1/_2X^2$. Для печати таблиц значений обеих функций потребуются дополнительные команды вывода. Соответствующая этим требованиям блок-схема алгоритма показана на рис. 5.7. Следует обратить внимание, что имеет место семикратное повторение выполнения четырех команд: присвоение переменной Y значения X^2; присвоение переменной Z значения ${}^1/_2X^2$; вывод на печать значений X и Y; вывод на печать значений X и Z. Эти команды можно объединить в подпрограмму с целью обращения к ней по мере необходимости. Рис. 5.8 демонстрирует такую организацию блок-схемы алгоритма.

Использование подпрограммы сокращает длину программы. Блок, символизирующий команду «Переход к подпрограмме», является точкой передачи управления подпрограмме. Блок-схема алгоритма подпрограммы начинается блоком «Вход», за которым следуют блоки присвоения переменной Y значения X^2, переменной Z значения ${}^1/_2X^2$, вывода на печать значений X и Y, а также X и Z. Завершает блок-схему блок «Возврат», обозначающий возврат управления в ту точку программы, из которой происходило обращение к подпрограмме, т.е. на вход блока, расположенного после блока «Переход к подпрограмме». На рис. 5.9 схематически изображено семикратное обращение к подпрограмме в процессе выполнения главной программы.

Данный пример использования подпрограммы отличается простотой решения. Возможны и другие подходы к этой задаче. Здесь же преследуется единственная цель – продемонстрировать достоинства применения подпрограммы.

Из одной программы можно производить обращение к нескольким подпрограммам. На рис. 5.10 показано схематическое изображение алгоритма программы, в которой осуществляется двукратный вызов двух подпрограмм A и B. Согласно этой схеме, порядок действий можно описать следующим образом:

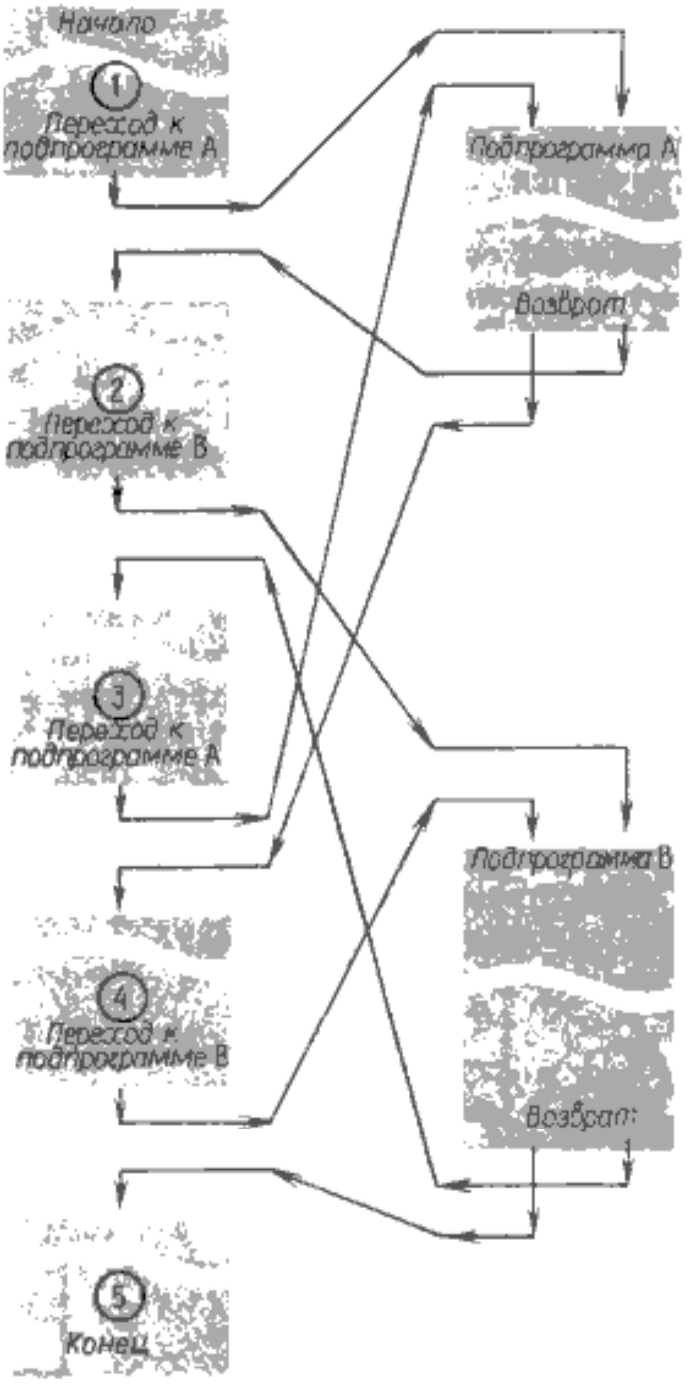


Рис. 5.10. Схема двукратного обращения к двум подпрограммам из главной программы

главная программа (часть 1),
подпрограмма A;
главная программа (часть 2);
подпрограмма B;
главная программа (часть 3);
подпрограмма A;
главная программа (часть 4);
подпрограмма B;
главная программа (часть 5).

Хотя в рассмотренных примерах начало и конец всех действий являются функцией главной программы, это не является обязательным правилом. В некоторых случаях выполнение завершается по команде

ОСТАНОВ в подпрограмме, и возврата в главную программу не происходит.

Одна подпрограмма может обращаться к другой. Такое построение подпрограмм называется *вложением*. Двухуровневое вложение демонстрирует рис. 5.11: главная программа обращается к подпрограмме A, которая в свою очередь обращается к подпрограмме B. На рис. 5.12 показана организация трехуровневого вложения. Команды главной программы разделены на *пять частей. Части 1–4 завершаются командами перехода к подпрограммам A*, B и C. Подпрограммы A и B в свою очередь содержат команды перехода к подпрограммам: подпрограмма A вызывает подпрограмму B, а последняя – подпрограмму C. Это – пример трехуровневого вложения подпрограмм. Порядок действий, соответствующий схеме, приведенной на рис. 5.12, имеет следующий вид:

главная программа (часть 1);
подпрограмма A (часть A1);
подпрограмма B (часть B1);
подпрограмма C
подпрограмма B (часть B2);
подпрограмма A (часть A2);
главная программа (часть 2);
подпрограмма C;
главная программа (часть 3);
подпрограмма B (часть B1);
подпрограмма C;
подпрограмма B (часть B2);
главная программа (часть 4);
подпрограмма C;
главная программа (часть 5).

Здесь имеет место организация одно-, двух- и трехуровневого вложения подпрограмм. Кроме того, это – пример обращения к группе подпрограмм.

Глубина допустимого уровня вложения подпрограмм зависит от типа вычислительной машины и используемого языка программирования. Большинство современных микропроцессоров и языков программирования допускает многоуровневое вложение.

Как можно было убедиться, исчользова-

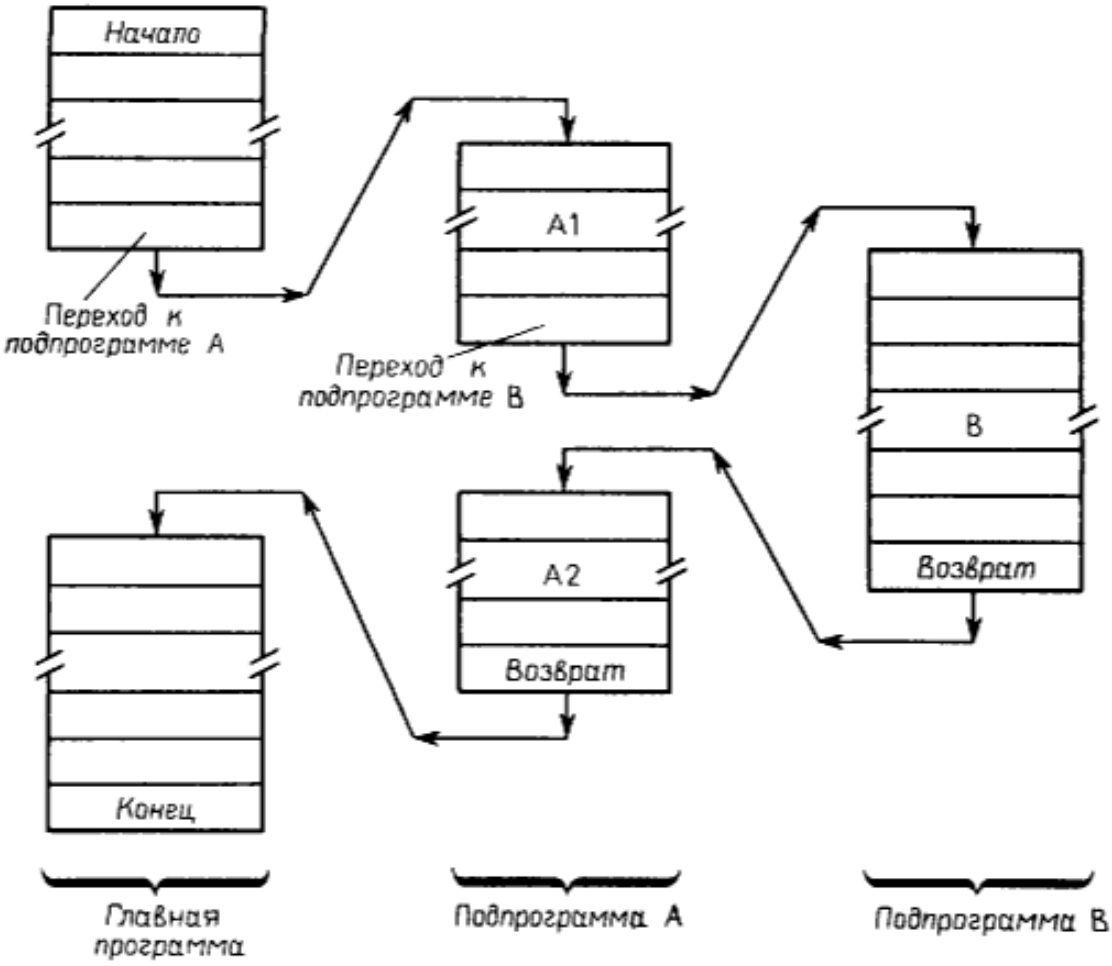


Рис. 5.11. Двухуровневое вложение подпрограмм.

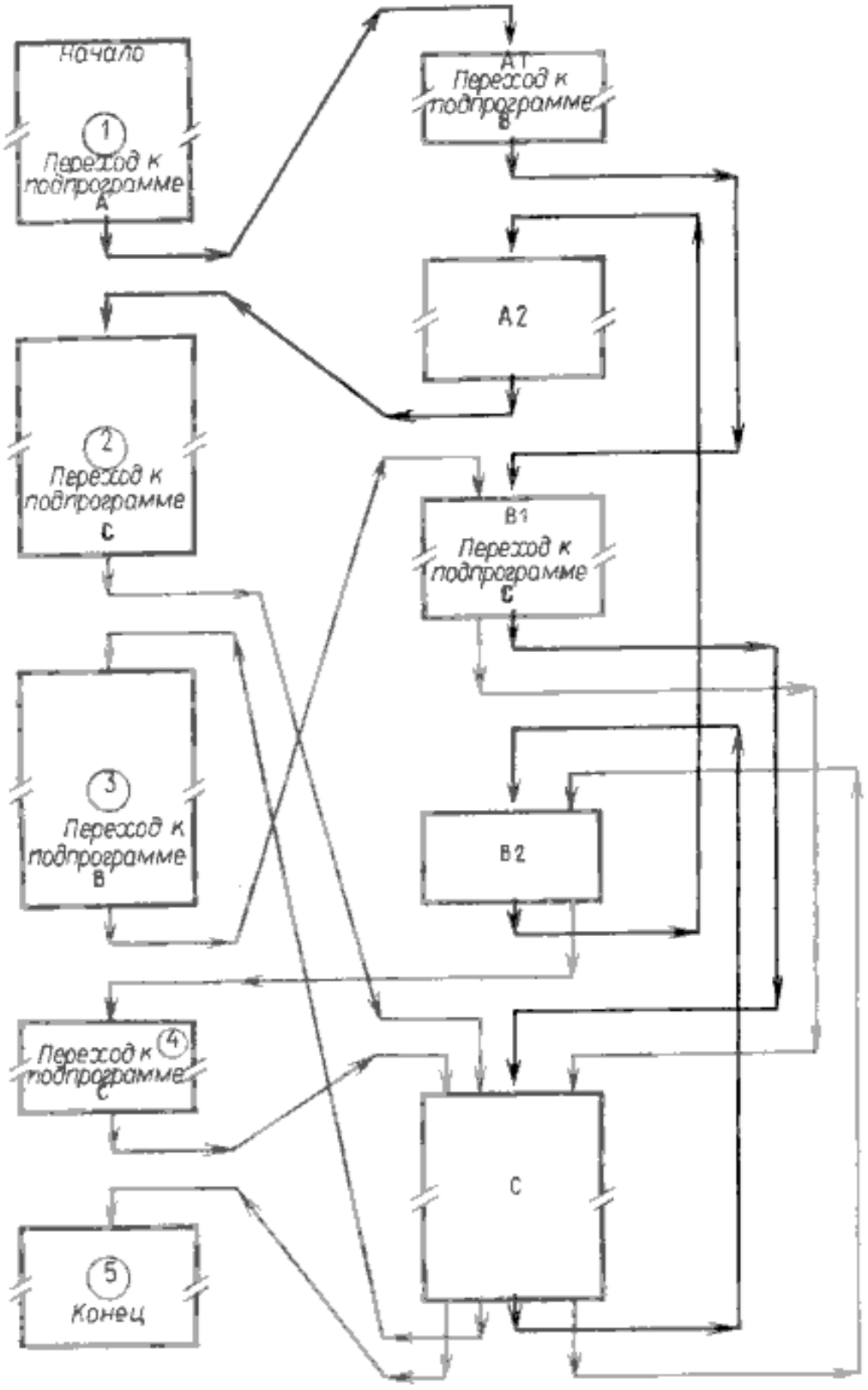


Рис. 5.12. Многоуровневое вложение подпрограмм.

ние подпрограмм – не слишком обременительная работа для программиста. Ему не следует беспокоиться о том, чтобы своевременно произошло возвращение из подпрограммы. Эту «заботу» берет на себя микропроцессор или язык программирования. Как это происходит на практике, будет описано в последующих главах.

Задания для самопроверки

13. Каково назначение подпрограммы: а) обеспечивать вложение программных модулей, б) сокращать длину программ, в) предоставлять возможность выполнения операций входа и возврата или г) позволять одновременное использование группы команд вывода на печать?

14. Укажите, с какой целью используется вложение подпрограмм: а) для сокращения длины подпрограммы; б) для уменьшения количества переходов к подпрограмме; в) чтобы подпрограмма могла производить обращение к широко используемым процедурам или г) для реализации всех перечисленных действий?

15. Использование подпрограммы часто позволяет сократить размер программы. Можно ли сказать, что такое сокращение равно а) 10%, б) 1/7, в) очень малой величине, однако делает программы удобочитаемыми или г) величине, зависящей от числа обращений к подпрограмме?

16. Имеется программа, описываемая следующей очередностью выполнения ее частей:

главная программа (часть 1);
подпрограмма А (часть А1);
подпрограмма А (части А1 и А2);
главная программа (часть 2).

а) Составьте схему переходов между главной программой и подпрограммами.

б) Что можно сказать об обращении подпрограммы к другим частям программы?

5.4. Языки программирования

Язык программирования – это набор команд и правил их применения для описания алгоритма решения задачи с целью составления программы работы вычислительной машины, выполняющей это решение. Подобно любым другим языкам, языки программирования располагают определенным словарным запасом и правилами его использования. Эти правила поясняют, как с помощью команд заставить вычислительную машину выполнять то, что нам необходимо.

Разные языки программирования имеют различную степень сходства с разговорным (естественным) языком. Чем ближе язык программирования к естественному языку, тем больше сходство команд языка программирования с предложениями разговорного языка. Например, написанное на *машинном языке* двоичное число 0100 1111 может выполнять роль команды очистки аккумулятора. Ничем эта команда не напоминает соответствующего распоряжения на таком языке, как, например, английский. Та же команда, выраженная на языке ассемблера, имеет вид CLA A. Такую запись можно рассматривать как аббревиатуру глагола английского языка clear (очищать). На языке высокого уровня БЕЙСИК переменной А можно присвоить нулевое значение посредством команды LET A = 0 (ПУСТЬ A = 0). В этом случае между командой языка высокого уровня и предложением английского языка уже много общего.

Но, если языки программирования с высокой степенью сходства с разговорным языком значительно проще для понимания, почему мы пользуемся языками низкого уровня? Это объясняется двумя причинами. Во-первых, при использовании языков высокого уровня по сравнению с языками низкого уровня требуется больше машинных команд для выполнения той же самой работы. Например, для реализации команды LET A = 0 языка БЕЙСИК может понадобиться 100 и более машинных команд. В то же время двоичное слово 0100 1111 является машинной командой (единственной!), осуществляющей очистку (сброс) аккумулятора. При работе с командами языка высокого уровня требуется больший объем памяти для хранения данных и большее время для выполнения. Следовательно, программы на машинном языке более эффективны, чем решающие те же задачи программы на языках высокого уровня. Во-вторых, использование языков высокого уровня возможно только при работе с вычислительными системами сравнительно сложной архитектуры. Так, для ведения диалога между программистом и машиной требуется специальное устройство ввода-вывода. Для выполнения за сравнительно короткое время программы, написанной на языке высокого уровня, необходимы мощный процессор и память значительного объема.

Несмотря на большое разнообразие языков программирования различных уровней, программы на этих языках подлежат преобразованию в последовательности двоичных машинных команд, прежде чем они могут быть выполнены вычислительной машиной. Программируя непосред-

ственно на машинном языке, можно вводить двоичные слова прямо в память. При использовании языка высокого уровня требуется промежуточный этап преобразования операторов языка высокого уровня в машинные команды. А для этого необходима более мощная вычислительная система, способная выполнять подобное преобразование, называемое *трансляцией*.

Программирование на машинном языке является некоторым общим программным уровнем для всех вычислительных машин. Машина определенного типа имеет свой набор машинных команд, и, следовательно, машинные коды ЭВМ одного типа неприемлемы для использования на ЭВМ других типов.

Программирование на машинном языке применяется только при составлении очень коротких микропроцессорных программ. Если длина программы должна быть больше, чем несколько байтов, программирование на машинном языке становится трудоемким. В таком случае приходится помнить начальный адрес программы и адреса всех команд, чтобы иметь возможность при необходимости обратиться к той или иной команде. И если обнаруживается, что какая-либо команда пропущена, то приходится переписывать всю программу.

При работе с микропроцессорами наиболее распространенным является программирование на *языке ассемблера*, в котором используется сокращенное написание английских слов (мнемоническое обозначение) для наименования каждой двоичной команды. Так, выше упоминалась команда CLA этого языка.

Запись программы на языке ассемблера ведется с использованием символических адресов, т.е. вместо числовых значений адресов используются имена. Начало программы должно быть «привязано» к определенному числовому адресу, относительно которого и ведется отсчет значений символических адресов.

На рис. 5.13, *а* показан пример исходной программы на языке ассемблера, на рис. 5.13, *б* – так называемый листинг этой программы, создаваемый в процессе ассемблирования исходной программы. Данная программа производит очистку всех областей (байтов) памяти с адресами от 000.100_8 до 000.377_8. Ссылка на действительный адрес памяти встречается в исходной программе только один раз, а именно в первой команде ORG 000000_8, информирующей ассемблер (транслятор исходной программы в двоичные машинные коды) о необходимости поместить начало программы в область памяти с адресом 000.000_8. В дальнейшем тексте программы ссылки на адреса областей памяти носят символический характер в виде таких имен, как AGAIN и START.

На рис. 5.13, *б* показаны действительные адреса областей памяти и машинные коды команд, выраженные в восьмеричной системе счисления. (При записи восьмеричных адресов точка используется для отделения цифр младшего и старшего байтов.) Символическому адресу AGAIN ассемблер присваивает фактический адрес 000.003_8.

Программу на языке ассемблера составляют с помощью служебной программы, называемой *редактором*. Последняя помогает программисту легко и без ошибок записывать команды языка ассемблера, адреса и данные. Если ошибки и возникают, то редактор позволяет их исправить без особых затруднений.

Написанная и откорректированная с помощью редактора программа может ассемблироваться. *Ассемблер* – служебная программа, преобразующая символические имена (сокращенные английские слова) и символические адреса в команды в машинном коде и числовые адреса. Процесс редактирования – ассемблирования схематически изображен на рис. 5.14. Этот процесс носит итерационный характер; после корректировки всех ошибок его повторение завершается.

Часто алгоритм решения задачи записывается в виде нескольких коротких программных модулей. После ассемблирования каждого программного модуля для объединения модулей используется программа *редактор связей*. Редактор связей определяет каждой машинной команде свое место в памяти и обеспечивает программным модулям возможность обращения друг к другу.

На языке ассемблера можно писать большие и сложные программы, не усту-

Исходная программа			Комментарии
	ORG	000000A	ПРОГРАММА НАЧИНАЕТСЯ С 0^8
START	LXI	B,000100A	ЗАГРУЗКА В ВС 000100_8
AGAIN	XRA	A	УСТАНОВКА АККУМУЛЯТОРА В 0
	STAX	B	ЗАПИСЬ СОДЕРЖИМОГО АККУМУЛЯТОРА В ПАМЯТЬ
	INX	B	АДРЕСАЦИЯ СЛЕДУЮЩЕЙ ОБЛАСТИ ПАМЯТИ
	MOV	A,C	ЗАПИСЬ СОДЕРЖИМОГО С В А
	CPI	377Q	ПРОВЕРКА, ДОСТИГЛО ЛИ СОДЕРЖИМОЕ С ЗНАЧЕНИЯ 377_8
	JZ	AGAIN	НЕТ. ПРОДОЛЖЕНИЕ
	HLT		ДА. ОСТАНОВ
	END	START	

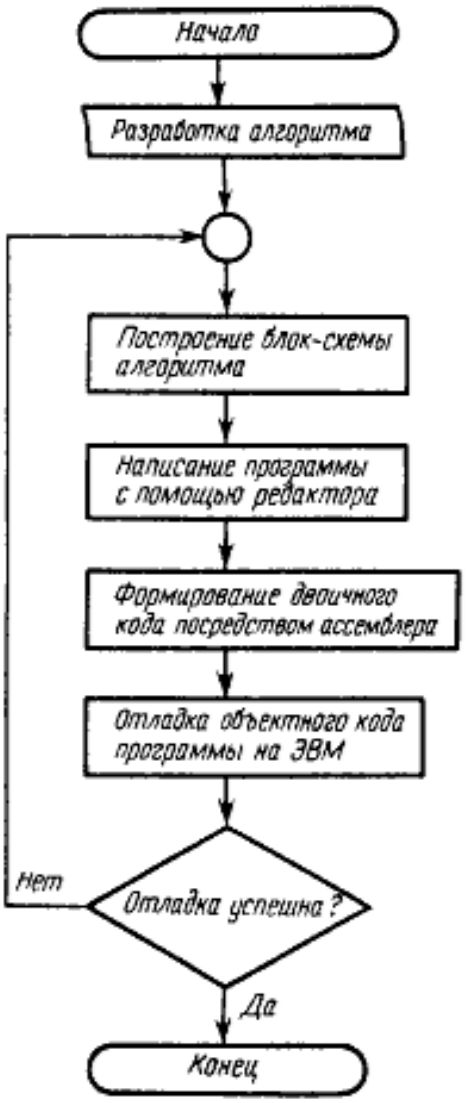
а

Восьмеричный адрес	Восьмеричный машинный код	Исходная программа			Комментарии
000.000			ORG	000000A	ПРОГРАММА НАЧИНАЕТСЯ С 0_8
000.000	001 100 000	START	LXI	B, 000100A	ЗАГРУЗКА В ВС 000100_8
000.003	257	AGAIN	XRA	A	УСТАНОВКА АККУМУЛЯТОРА В 0
000.004	002		STAX	B	ЗАПИСЬ СОДЕРЖИМОГО АККУМУЛЯТОРА В ПАМЯТЬ
000.005	003		INX	B	АДРЕСАЦИЯ СЛЕДУЮЩЕЙ ОБЛАСТИ ПАМЯТИ
000.006	171		MOV	A,C	ЗАПИСЬ СОДЕРЖИМОГО С В А
000.007	376 377		CPI	377 Q	ПРОВЕРКА, ДОСТИГЛО ЛИ СОДЕРЖИМОЕ С ЗНАЧЕНИЯ 377_8
000.011	312 003 000		JZ	AGAIN	НЕТ. ПРОДОЛЖЕНИЕ
000.014	166		HLT		ДА. ОСТАНОВ
000.015	000		END	START	

00011 Statements Assembled
03475 Bytes Free
No Errors Detected

б

Рис. 5.13. Ассемблирование исходной программы: *а*) исходная программа, в которой использовано мнемоническое обозначение языка ассемблера (текст комментария ассемблер игнорирует); *б*) листинг ассемблирования с восьмеричными кодами адресов памяти и машинных команд.



5.14. Блок-схема алгоритма создания программы на языке ассемблера.

пающие по эффективности машинным кодам, потому что одной команде языка ассемблера соответствует одна машинная команда. В гл. 6 и других главах описываются программы на языке ассемблера с соответствующим мнемоническим обозначением команд. Объясняется, как преобразовать команды языка ассемблера в машинные команды, которыми оперирует микропроцессорная система, т.е., по существу, излагается методика ассемблирования программ вручную, без участия ЭВМ.

Перевод текста программ с языков высокого уровня на язык машинных команд может осуществляться двумя способами: путем *интерпретации* или *трансляции*; соответствующая служебная программа-переводчик называется *интерпретатором* или *транслятором*. Типичным примером языка высокого уровня является язык БЕЙСИК (BASIC – Beginner's All-puprose Symbolic Instruction Code – язык символического кодирования общего назначения для начинающих). Для перевода команд этого языка (похожих на предложения английского языка) в машинные команды обычно используется интерпретатор, представляющий собой довольно сложную программу.

Интерпретатор работает в процессе выполнения программы. Каждая строка программы на языке высокого уровня преобразуется в машинные команды непосредственно перед ее выполнением. Анализируя содержимое строки программы, интерпретатор расчленяет ее на части, которые преобразуются в машинные команды с помощью набора подпрограмм. После преобразования в машинный код строка исходной программы выполняется. По завершении обработки одной строки программы интерпретатор начинает подобный процесс преобразования другой строки.

Из анализа работы интерпретатора следует, что он использует очень много подпрограмм. Эти подпрограммы совместно со служебной программой синтаксического анализа команд языка высокого уровня, а также команды программы на языке высокого уровня и исходные данные размещаются в памяти. Следовательно, для работы интерпретатора требуется микропроцессорная система с большой памятью. Так, для интерпретатора языка БЕЙСИК при работе на 8-разрядной микро-ЭВМ необходимо располагать от 12 до 24К байт памяти.

Основное достоинство работы с интерпретатором заключается в возможности программировать на языке высокого уровня. Можно легко решать сложные математические задачи, не прибегая к записи большого количества машинных кодов. Например, используя язык БЕЙСИК, можно обращаться к подпрограммам вычисления логарифмов, синусов, извлечения квадратных корней и определения значений других математических функций с помощью только одного оператора. Для достижения этих же целей, применяя язык ассемблера или машинных команд, требуется от 1000 до 4000 команд. Интерпретатор

языка БЕЙСИК часто используют при работе с небольшими вычислительными системами на базе микро-ЭВМ для проведения научных исследований, заменяя менее пригодные в подобных случаях мини-ЭВМ. Многие микро-ЭВМ общего назначения располагают интерпретатором языка БЕЙСИК. То же можно сказать и о микро-ЭВМ индивидуального пользования. Существуют и другие языки, использующие интерпретаторы, однако БЕЙСИК является в этом отношении самым распространенным языком.

Как ранее упоминалось, программа на языке высокого уровня может быть преобразована в программу на машинном языке также и посредством компилятора. Последний во многом похож на ассемблер. Но если ассемблер преобразует в машинный код команды на языке ассемблера, то компилятор производит те же действия над командами на языках высокого уровня. По завершении формирования программы на языке машинных команд, называемой объектным кодом, компилятор больше не требуется. Для последующего выполнения объектного кода необходим меньший объем памяти, чем тот, который нужен для той же самой программы и интерпретатора, присутствующего в памяти при выполнении программы. Но если компилятор обеспечивает экономию памяти ЭВМ, почему бы не пользоваться им во всех случаях? Компилятору не всегда отдается предпочтение, поскольку он как программа значительно больше интерпретатора. Компиляторы столь значительны по размеру, что в микро-ЭВМ вообще не используются. Для них оказывается недостаточным объем памяти даже некоторых мини-ЭВМ малого и среднего размеров. Только большие ЭВМ всегда позволяют использовать компиляторы, размер которых зависит от их сложности. Отметим, что даже сравнительно простые из них представляют собой весьма обширные программы.

Не следует, однако, недооценивать значения компиляторов. Они часто применяются при создании объектных модулей для микропроцессоров, что позволяет последним выполнять сложные и значительные по объему математические расчеты. Подобные расчеты значительно проще описать на языке высокого уровня, чем на языке ассемблера. Написанная на языке высокого уровня, такая программа слишком велика для выполнения на микро-ЭВМ. В то же время, используя компилятор, работающий на большей ЭВМ, можно сформировать объектный модуль, достаточно короткий для прогона на микро-ЭВМ. Выполнение программ подобной сложности на микро-ЭВМ с помощью интерпретатора могло бы оказаться невозможным, поскольку при этом в памяти машины должен находиться и интерпретатор, занимающий значительный объем.

У интерпретаторов и ассемблеров имеется одно важное достоинство – наличие *встроенных редакторов*, благодаря чему, используя одну и ту же служебную программу ассемблера или интерпретатора можно не только составлять текст исходной программы, но и выполнять ее на следующем этапе. К недостаткам ассемблеров и интерпретаторов можно отнести их зависимость от модели микропроцессора, на работу с которым они ориентированы. Вот почему, прежде чем выбрать микропроцессор для программирования на языке высокого уровня, необходимо выяснить, каким программным обеспечением он располагает.

Среди компиляторов языков высокого уровня в настоящее время наиболее распространенными являются компиляторы языков ФОРТРАН и ПАСКАЛЬ. Один из давно используемых языков программирования ФОРТРАН ориентирован на выполнение научно-технических расчетов и отличается наибольшей степенью стандартизации. Хотя этот язык сравнительно сложен и несколько старомоден для современных средств вычислительной техники, он все еще продолжает оставаться одним из самых распространенных языков программирования. ПАСКАЛЬ относится к современным языкам; при работе с ним, как правило, используется компилятор. Он также ориентирован на проведение научно-технических расчетов и находит широкое применение при программировании для микропроцессорных систем. Свое название этот язык получил в честь французского математика Блеза Паскаля (1623–1662). По

сравнению с ФОРТРАНом язык ПАСКАЛЬ проще в использовании и лучше согласуется с методикой программирования на языке ассемблера или машинном языке.

В настоящее время получили широкое распространение *системы разработки и отладки программ* для микропроцессоров. Это небольшие системы общего назначения, реализованные на базе микро-ЭВМ, которые предоставляются в распоряжение пользователя для написания и отладки программ, подлежащих выполнению на том или ином микропроцессоре. Как правило, такие программы пишут на языке ассемблера, однако многие подобные системы допускают программирование на языках БЕЙСИК или ПАСКАЛЬ. Кроме программирования и отладки программ эти системы часто используются для поиска неполадок в сложных микропроцессорных системах. Более подробно вопрос, связанный с системами разработки и отладки программ для микропроцессоров, описывается в гл. 14.

Несмотря на разнообразие средств программирования работы микропроцессора – язык машинных команд, язык ассемблера, языки высокого уровня, – принципы программирования остаются одними и теми же. Блок-схема алгоритма решения той или иной задачи может быть записана на любом из этих языков.

В дальнейшем основное внимание уделяется программированию на машинном языке и языке ассемблера. Описание средств программирования на языке высокого уровня, как правило, не требует знакомства с архитектурой микропроцессорной вычислительной системы. Однако эффективное программирование на машинном языке или языке ассемблера, при котором отсутствуют ошибки, невозможно без знания основ этой архитектуры.

Задания для самопроверки

17. Предположим, что необходимо написать программу из 22 команд, производящую очистку всех областей памяти. Какие средства программирования предпочтительнее использовать для этой цели: а) машинный язык, б) язык ассемблера, в) язык высокого уровня, подлежащий интерпретации, или г) язык высокого уровня, подлежащий компилированию?

18. Вам предоставляется в индивидуальное пользование микро-ЭВМ. Требуется написать программу вычисления корней квадратного уравнения согласно выражению $(-b \pm \sqrt{b^2 - 4ac})/2a$. Какой из языков, перечисленных в п. 17, вы бы предпочли?

19. Требуется разработать программное обеспечение сравнительно небольшой системы автоматического управления на базе микро-ЭВМ, которая занимает 14К байт памяти. Это программное обеспечение должно включать в свой состав несколько подпрограмм, занимающих небольшой объем памяти и выполняемых за сравнительно короткое время. На каком из языков программирования, перечисленных в п. 17, вы предпочли бы создавать программное обеспечение такой системы?

20. Предположим, вы обслуживаете систему анализа и отображения информации, разработанную на базе микро-ЭВМ и используемую в медицине. Обработка данных связана с выполнением сравнительно большого объема математических расчетов. Разработку программ предполагается проводить на большой мини-ЭВМ с использованием одного из языков, перечисленных в п. 17. Какой язык вы могли бы порекомендовать для этой цели?

21. Составьте краткие письменные ответы на вопросы, поставленные в пп. 17–20.

Упражнения

5.1. Требуется определить время свободного падения некоторого предмета с определенной высоты на землю. Можно ли использовать вычислительную машину для изменения алгебраического выражения $s = gt^2/2$ или с целью получения различных значений s придется задавать различные значения параметра t? Поясните ответ.

5.2. Что является «исходным материалом» для написания программы?

5.3. При подготовке программы к решению на ЭВМ алгоритм записывается в виде последовательности команд. Как называется такая последовательность?

5.4. Бóльшую часть времени отладка программы производится на той же машине, которая и выполняет эту программу. В процессе отладки осуществляются поиск и исправление ошибок.

Какие ошибки удается устранить – логические или синтаксические? Обоснуйте ваш ответ.

5.5. Что такое алгоритмизация задачи, подлежащей решению?

5.6. Для какой из нижеперечисленных целей предназначена блок-схема алгоритма: для последующей записи текста программы, разработки алгоритма решения задачи, документирования программы или для всех указанных целей?

5.7. Что обозначает на блок-схеме алгоритма блок, имеющий форму прямоугольника: обработку данных, проверку условия с последующим принятием решения, межблочное соединение или начало (окончание) хода вычислений?

5.8. Что обозначает на блок-схеме алгоритма кружок: обработку данных, проверку условия с последующим принятием решения, межблочное соединение или начало (окончание) хода вычислений?

5.9. Что обозначает на блок-схеме алгоритма блок, имеющий овальную форму: обработку данных, проверку условия с последующим принятием решения, межблочное соединение или начало (конец) хода вычислений?

5.10. Что обозначает на блок-схеме алгоритма блок, имеющий форму ромба: обработку данных, проверку условия с последующим принятием решения, межблочное соединение или начало (конец) хода вычислений?

5.11. Используя четыре стандартных блока, составьте блок-схему алгоритма процесса разработки алгоритма решения задачи, составления программы, ее отладки и выполнения.

5.12. В чем заключаются достоинства использования вложенных подпрограмм при программировании?

5.13. Должна ли подпрограмма быть короче, чем главная программа?

5.14. Должна ли подпрограмма вызываться более одного раза?

5.15. Дайте краткое объяснение назначению подпрограммы.

5.16. Объясните, в чем состоит различие между ассемблером и компилятором.

5.17. Положим, что в вашем распоряжении имеются три программы решения одной и той же задачи: на языке ассемблера, ФОРТРАНе и БЕЙСИКе. Расположите их в порядке убывания объема памяти, который необходим для их размещения в ЭВМ с целью последующего выполнения.

5.18. В процессе отладки программы выявляются ошибки в исходной программе. Какие средства для этого необходимы: редактор, ассемблер, интерпретатор, машинные коды или все перечисленные средства?

5.19. Предположим, что программа, написанная на языке высокого уровня, выполняется машиной по мере чтения очередного оператора. Какой из нижеперечисленных языков программирования обеспечивает такое выполнение программы: редактора, ассемблера, компилятора, интерпретатора, машинных кодов или все перечисленные языки вместе?

5.20. В какую программу в конечном счете преобразуется программа, написанная на любом из языков программирования: в программу редактора, ассемблера, компилятора, интерпретатора, машинных кодов или в совокупность всех перечисленных программных средств?

5.21. Предположим, что вы пишете программу на языке высокого уровня для ЭВМ модели 145 Системы 370 фирмы IBM с целью получения объектного модуля для вашей микро-ЭВМ. Каким средством программирования Системы 370 вы пользуетесь для этой цели: редактором, ассемблером, компилятором, интерпретатором, машинными кодами или всеми перечисленными?

5.22. Какой из указанных языков программирования обеспечивает перевод одной команды на этом языке в одну машинную команду: редактора, ассемблера, компилятора, интерпретатора, машинных кодов или все перечисленные языки?

5.23. Что можно назвать словарем языка программирования, и каким образом нам известно его назначение и сфера применения?

Ответы на вопросы заданий для самопроверки

1. а) Температура равна 20 С?
   б) Да. Выключить отопление и кондиционер и перейти к п. «а».
   в) Нет. Температура выше 20°С?
   г) Да. Включить кондиционер и перейти к п. «а».
   д) Нет. Включить отопление и перейти к п. «а».

2.

Положить X = 2	Положить X = 4
Вычислить $Y = X^2$	Вычислить $Y = X^2$
Вычислить $Z = X^3$	Вычислить $Z = X^3$
Вычислить $U = \sqrt{X}$	Вычислить $U = \sqrt{X}$
Вычислить $W = \sqrt[3]{X}$	Вычислить $W = \sqrt[3]{X}$
Вывести на печать X, Y, Z, U, W	Вывести на печать X, Y, Z, U, W
Положить X = 3	Положить X = 5
Вычислить $Y = X^2$	Вычислить $Y = X^2$
Вычислить $Z = X^3$	Вычислить $Z = X^3$
Вычислить $U = \sqrt{X}$	Вычислить $U = \sqrt{X}$
Вычислить $W = \sqrt[3]{X}$	Вычислить $W = \sqrt[3]{X}$
Вывести на печать X, Y, Z, U, W	Вывести на печать X, Y, Z, U, W

3. в. 4. а. 5. б.

6. ЭВМ способна выполнять только ту последовательность команд, которая ей задана. Если решение задачи вам неизвестно, то вы не можете за-

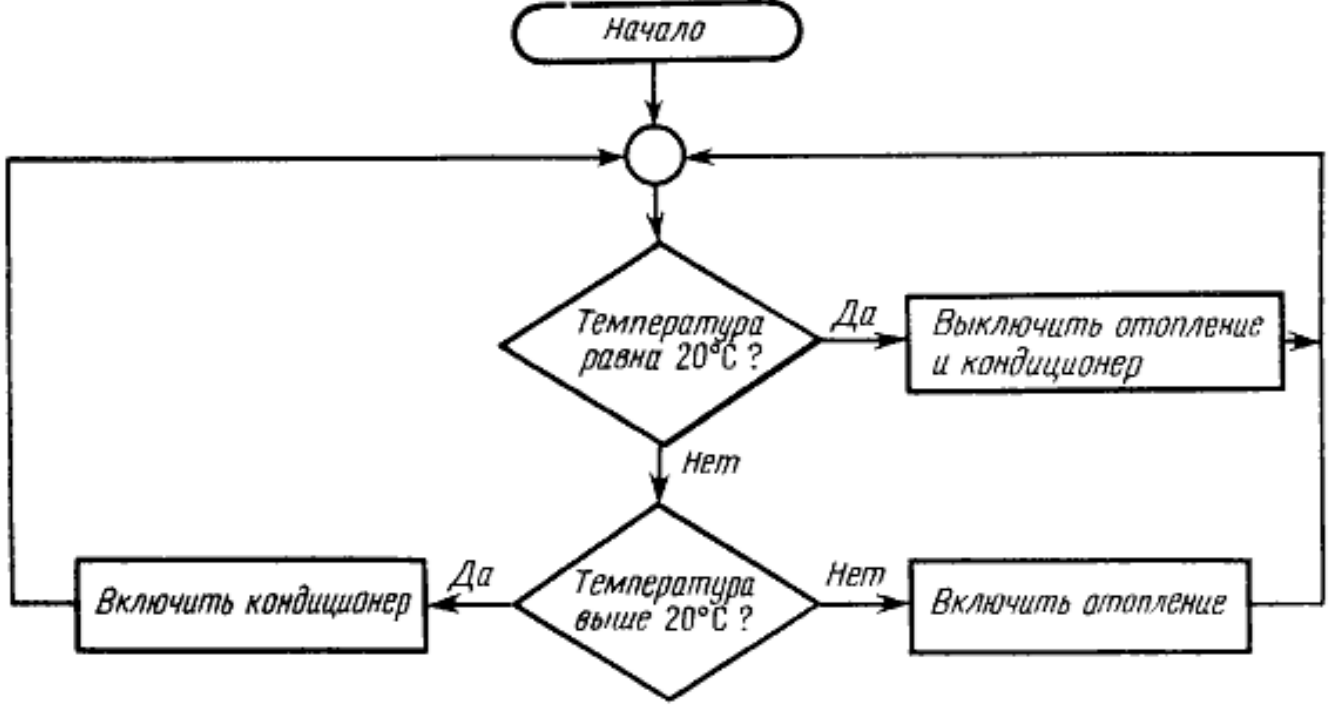


Рис. 5.15. Блок-схема алгоритма решения задачи к ответу на вопрос п. 1 заданий для самопроверки.

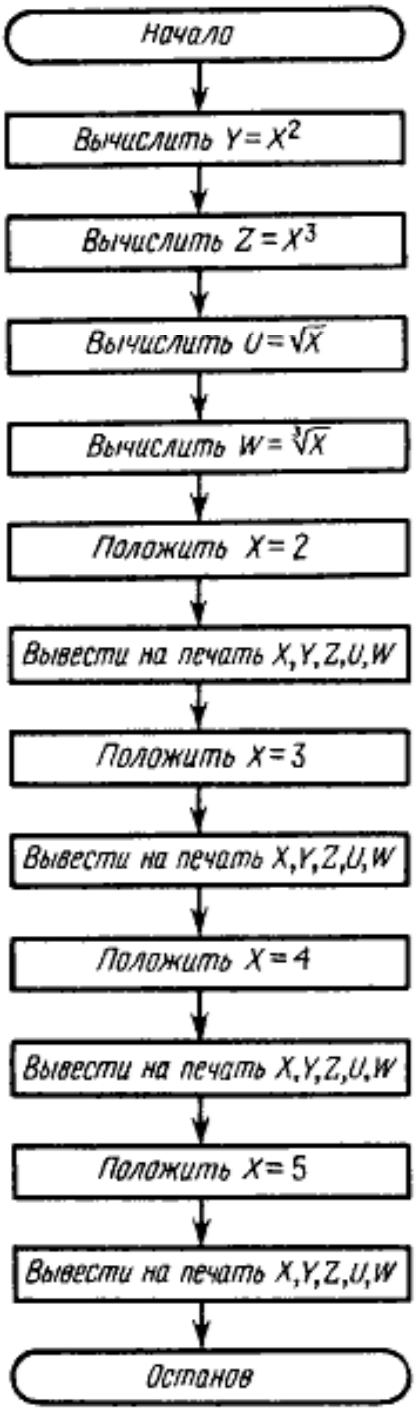


Рис. 5.16. Блок-схема алгоритма решения задачи к ответу на вопрос п. 2 заданий для самопроверки.

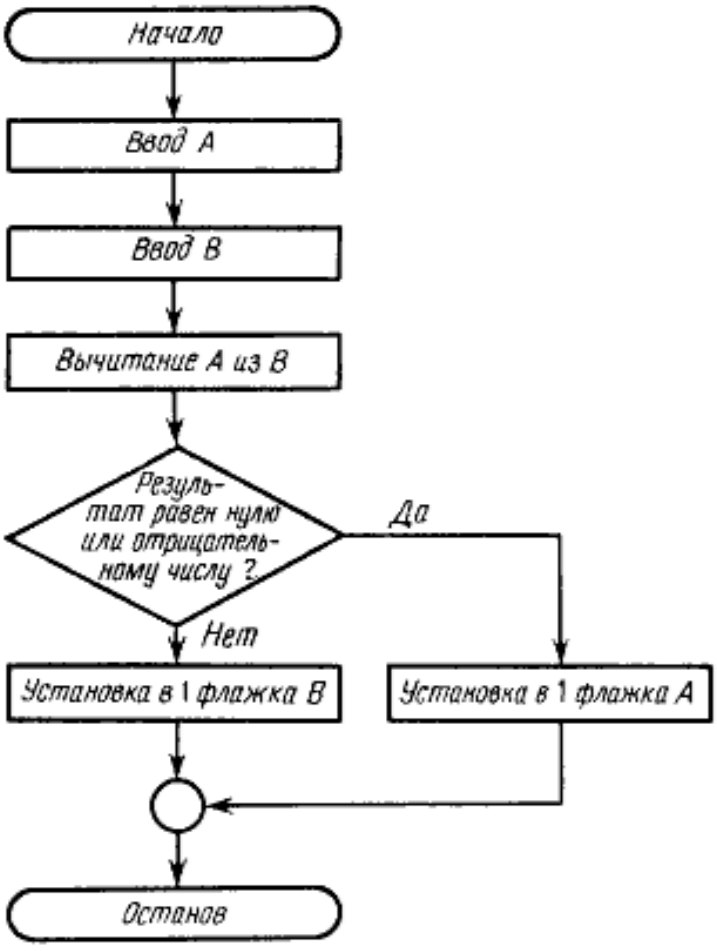


Рис. 5.17. Блок-схема алгоритма задачи к ответу на вопрос п. 9 заданий для самопроверки.

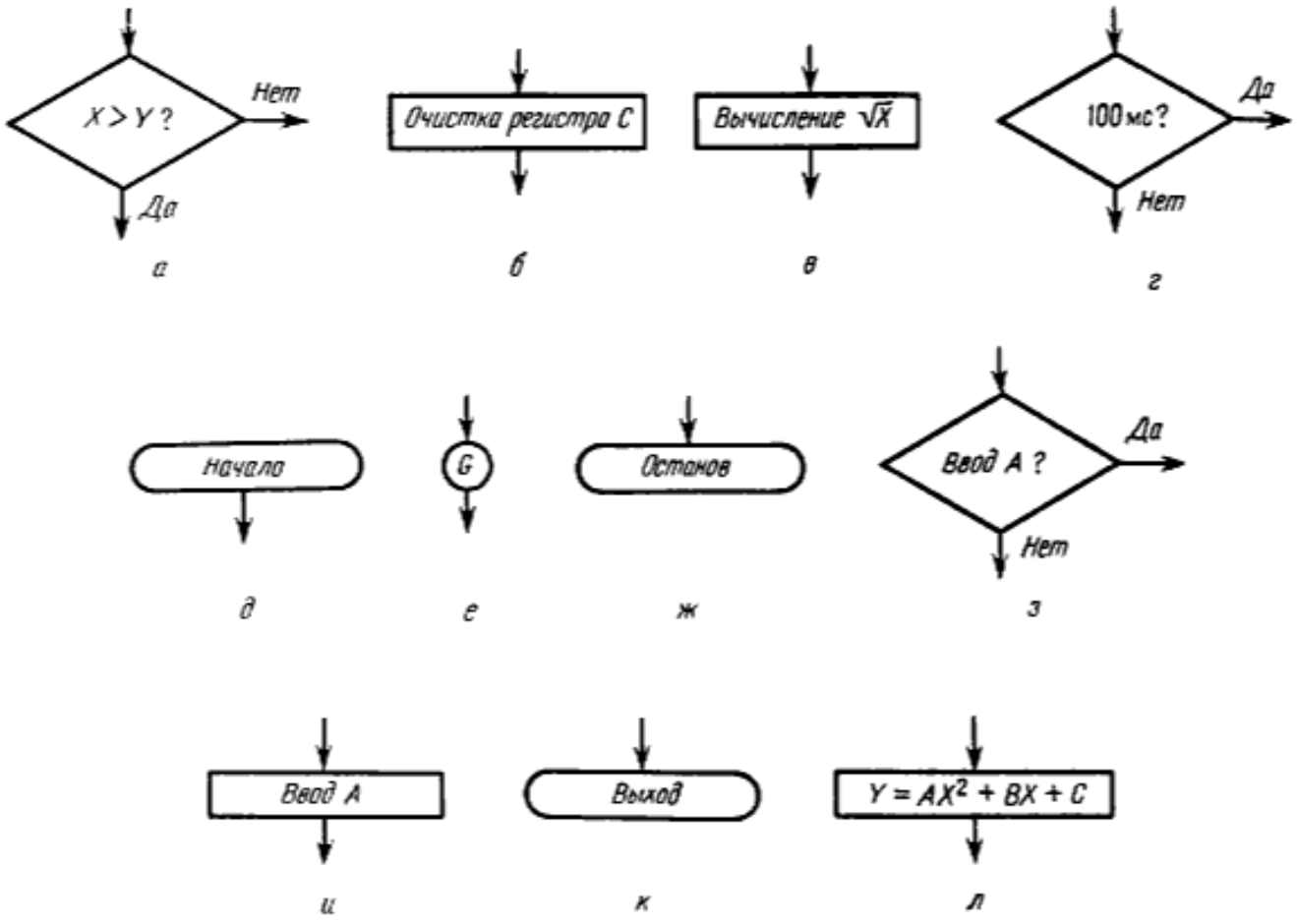


Рис. 5.18. Ответы на вопросы п. 12 заданий для самопроверки.

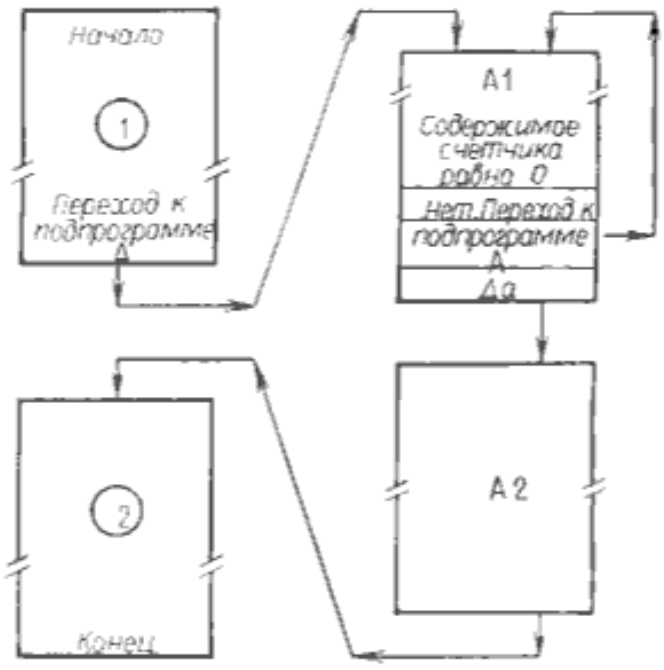


Рис. 5.19.

дать для машины соответствующую последовательность команд, и, следовательно, такая задача машине не под силу.

7. См. рис. 5.15 и 5.16.

8. На рис. 5.15 показана блок-схема алгоритма, содержащая блоки проверки условий с последующим принятием решений, а также петли обратной связи. Указанные узлы предназначены для определения температуры. Блок-схема алгоритма (рис. 5.16) состоит только из расположенных друг за другом блоков описания операций, последовательное выполнение которых приводит к желаемому результату.

9. См. рис. 5.17.

10. Направление хода вычислений указывают стрелки на линиях, соединяющих блоки.

11. Программа начинается присвоением содержимому регистра В значения, равного 1000. Затем периодически осуществляется проверка, не стало ли содержимое регистра В равно 2001. Если ответ отрицательный, то содержимое этого регистра получает единичное положительное приращение, после чего опять выполняется проверка. Когда

содержимое регистра B становится равным 2001, работа программы заканчивается. Такую программу называют подпрограммой или процедурой задержки, потому что микропроцессору приходится затратить определенное время на выполнение этого циклического процесса. В данном случае измеренная в относительных единицах задержка равна 2001 — 1000 = 1001 (раз).

12. См. рис. 5.18.

13. б. 14. в. 15. г.

16. а) См. рис. 5.19. б) Подпрограмма может вызвать себя.

17. а. 18. в. 19. б. 20. г.

21. *Ответ на задание № 17.*

Очень короткие программы, выполняемые непосредственно микропроцессором, наиболее удобны для записи на машинном языке. Однако если в распоряжении программиста находится весь арсенал средств программирования, он, вероятнее всего, отдаст предпочтение языку ассемблера при написании этой программы.

Ответ на задание № 18.

Для решения подобного уравнения придется обратиться к языку программирования высокого уровня. Если в распоряжении программиста имеется микропроцессорная вычислительная система, то, скорее всего, компилятор выбран не будет. Как правило, для подобных задач имеется возможность использовать интерпретатор.

Ответ на задание № 19.

Если требуется, чтобы подпрограммы были эффективны по своим характеристикам, для их записи, вероятнее всего, придется воспользоваться машинным языком или языком ассемблера. Однако вряд ли кому-нибудь захочется работать с программой на машинном языке размером в 14К байт.

Ответ на задание № 20.

Прибегать к услугам другой ЭВМ для разработки программного обеспечения данной машины приходится в тех случаях, когда необходим сравнительно сложный компилятор или ассемблер.

Для проведения математических расчетов вряд ли возникнет желание использовать язык ассемблера.

Глава 6. Команды микропроцессора

В этой главе дается определение понятия «команда микропроцессора»; показано, чем это двоичное слово отличается от других двоичных слов, хранимых в памяти; объясняется, из каких основных частей состоит команда; вводится мнемоническое обозначение команд и их составных частей. Читателю сообщается, что первая часть команды информирует микропроцессор о том, что́ делать; вторая часть указывает, откуда должны поступать данные для выполнения операции. Здесь же рассматриваются различные способы адресации и назначение каждого из них.

В заключение описывается небольшая микропроцессорная система, иллюстрирующая принципы построения типичной микро-ЭВМ и организацию ее команд, которые более подробно обсуждаются в других главах.

6.1. Что такое набор команд?

Команда микропроцессора – это такое двоичное слово, которое, будучи «прочитано» микропроцессором, заставляет последний выполнить определенные действия. Другие, отличные от команд двоичные слова подобных действий в микропроцессоре вызывать не могут. Большинство команд осуществляет пересылку или обработку данных, расположенных в памяти или в одном из регистров микропроцессора. Несколько команд предназначено для управления некоторыми вспомогательными функциями микропроцессора, поддерживающими необходимый режим его работы. Когда говорят о *наборе команд* микропроцессора, то подразумевают все его команды.

Длина команды как двоичного слова совпадает с длиной слова данных. Так, длина слова команды 8-разрядного микропроцессора равна 8 бит, а 16-разрядного микропроцессора – 16 бит. Однако команды могут иметь длину, равную не только одному, но также двум или трем словам. Следовательно, длина команды 8-разрядного микропроцессора может равняться 8, 16 или 24 бит. Назначение отдельных битов команды описывается ниже.

Для выполнения команда посылается в регистр команд, дешифратор и схемы управления, где она идентифицируется, в результате чего формируются сигналы, направляемые в другие части микропроцессора. С помощью этих сигналов выполняются операции, предписываемые командой.

Микропроцессор загружает команду в регистр команд в течение цикла выборки. В течение следующего за ним цикла выполнения микропроцессор декодирует команду и создает сигналы управления процессом выполнения операций этой команды.

Детальное рассмотрение команды микропроцессора показывает, что она должна содержать информацию двух видов. Во-первых, команда должна сообщать микропроцессору, что́ делать (выполнять сложение, очистку, пересылку, сдвиг и т. п.);

во-вторых, указывать адрес, т. е. местоположение обрабатываемых данных. Например, команды могут информировать микропроцессор о следующем: добавить к содержимому аккумулятора копию содержимого некоторой области памяти, очистить аккумулятор, переместить данные из регистра A в регистр B, выполнить сдвиг содержимого аккумулятора и т. п. Из приведенных примеров команд следует, что микропроцессор получает от команды информацию не только о том, что делать, но и о том, где находятся данные – объекты манипулирования.

Команда состоит из двух частей: *кода операции* (*КОП*) и *адреса*. Код операции сообщает микропроцессору, что́ делать; адрес указывает местоположение данных, участвующих в операции. Если длина команды составляет два или три слова, то первое из них – это код операции, а второе и третье – адрес. Из этого не следует делать вывод, что все команды длиной в одно слово являются безадресными. Подробнее этот вопрос анализируется при описании способов адресации.

В данной книге рассматривается восемь основных типов команд. На практике подавляющее число микропроцессоров располагает значительно большим количеством различных команд (как правило, не менее 50–75). Некоторые мощные микропроцессоры имеют набор из нескольких сотен команд.

Число команд значительно больше, чем число различных кодов операции, поскольку при формировании команды один и тот же код операции может использоваться при различных способах адресации. Например, большинство микропроцессоров использует код операции CLEAR (ОЧИСТКА, или СБРОС). Естественно задать вопрос: очистка чего? Ответ зависит от конкретного набора команд микропроцессора. Некоторые микропроцессоры располагают единственной подобной командой – командой очистки аккумулятора, другие – несколькими командами очистки, такими, как очистка аккумулятора A или B, очистка регистра A, B, C или D, очистка области памяти с адресом N. Как видно из последнего примера, сочетание одного кода операции с различными адресами может привести к созданию семи разных команд.

Задания для самопроверки

1. О чем информирует микропроцессор код операции: а) что делать, б) где выполнять предписываемые действия, в) что делать и где выполнять?

2. Чему равна длина команды микропроцессора по сравнению с длиной слова данных: а) такая же, б) в два раза длиннее, в) в три раза длиннее, или г) все приведенные утверждения справедливы?

3. О чем информирует микропроцессор адресная часть команды: а) о том, что делать, используя указанный код операции, б) где выполнять действия, предписываемые кодом операции, в) где находится команда или г) чему равна длина команды?

4. От каких нижеперечисленных параметров зависит число раз, которое определенный код операции используется в различных командах набора команд микропроцессора: а) от длины кода операции, б) от количества различных типов адресации, используемых кодом операции, в) от объема памяти микро-ЭВМ или г) от всех перечисленных параметров?

6.2. Мнемоническая форма записи команд

Команда микропроцессора – это двоичное число. Но даже 1-байтовое двоичное число трудно запомнить. Еще труднее «держать в голове» двоичные коды команд длиной в 2–3 байт. В предыдущих главах были рассмотрены восьмеричная и шестнадцатеричная системы счисления и их использование для более краткой записи длинных двоичных чисел. С помощью восьмеричных или шестнадцатеричных цифр можно было бы представлять и команды микропроцессора. Однако и в этом случае оставалась бы нерешенной основная проблема: что означает каждая команда, выраженная в подобной форме. Восьми- или шестнадцатеричные команды трудно было бы запомнить и отождествлять с их фактическим назначением.

Данная проблема решается путем применения мнемонического обозначения – сокращенной записи названия команды. Для этой цели обычно используются три буквы на-

звания операции, выполняемой командой. Например, мнемоническое обозначение команды очистки имеет следующий вид: CLA. Если микропроцессор содержит два аккумулятора (A и B), то команды их очистки могут записываться как CLA A и CLA B, где CLA – код операции, а A и B – адреса местоположения обрабатываемых данных. Если же команда оперирует числовыми данными или адресами областей памяти, то целесообразно использование чисел в адресной части команды. Например, код операции с мнемоническим обозначением JMP (JUMP – ПЕРЕХОД) требует указания адреса перехода. Подобная команда может иметь вид JMP 177756_8, где адрес выражен 6-разрядным восьмеричным числом, двоичный эквивалент которого $1111\ 1111\ 1110\ 1110_2$ есть адрес области памяти. Мнемоническое обозначение кода операции JMP легче запомнить, чем его восьмеричный эквивалент 303_8. Сочетание сокращенного буквенного обозначения кода операции с числовой формой записи адреса является одной из наиболее удобных форм записи команды.

В дальнейшем при написании программ мы будем пользоваться мнемонической записью, которая является не только удобной формой представления кода операции команды, но и составной частью последней при использовании языка ассемблера. Программа ассемблера преобразует мнемоническое обозначение кодов операции в соответствующие двоичные эквиваленты.

В данной книге введение каждой новой команды сопровождается ее мнемоническим обозначением с представлением соответствующих двоичных, восьмеричных и шестнадцатеричных эквивалентов. Поскольку мнемоническое обозначение кода операции – составная часть команды, то упомянутые эквиваленты, как правило, указываются и для адресной части команды. Руководства по эксплуатации микропроцессоров, выпускаемых промышленностью, содержат описания соответствующих наборов команд с указанием кодов операций как в виде мнемонической записи, так и в числовой форме. Потребность в такой документации возникает только тогда, когда составляют программу для работы микропроцессора, которым располагает разработчик.

Задания для самопроверки

5. Ранее было показано, что для обозначения кода операции команды перехода некоторого микропроцессора используются две формы записи – 303_8 и JMP. Какую из них называют мнемоническим обозначением? Поясните ответ.

6. Большинство мнемонических обозначений выражает суть действий, производимых командами. Что означают следующие мнемонические записи: CLA, NOP, MOV, HLT, INC, ADD, AND, DEC?

7. Длина некоторых мнемонических обозначений составляет не три, а четыре буквы. Означает ли это, что длина команды, код операции которой представляет мнемоническая запись, также изменяется? Поясните ответ.

8. Одной из функций программы ассемблера является преобразование исходной программы на языке ассемблера в программу на машинном языке. Можно предположить, что для выполнения указанного преобразования ассемблеру необходимо «просматривать» некоторую таблицу, содержащую мнемонические обозначения команд. Какого рода данные должны быть также включены в такую таблицу?

9. Каждой команде микропроцессор ставит в соответствие единственную комбинацию двоичных цифр. Можно ли утверждать, что каждая команда имеет лишь одно мнемоническое обозначение? Поясните ответ.

6.3. Способы адресации микропроцессора

В разд. 6.1 указывалось, что команда микропроцессора состоит из кода операции и адреса. Однако имеются команды и без адреса. Например, команде, приказывающей микропроцессору остановить работу, адрес не нужен. Но безадресных команд мало. Представьте себе, например, 8-разрядный микропроцессор с памятью объемом 65К. У программиста должна иметься возможность доступа к содержимому любой из 65 536 областей памяти. Для адресации к этим областям длина адресной части команды должна быть достаточной для раз-

мещения 16 бит. Конечно, двоичный код команды длиной 8 бит не может включать в себя код операции и 16-битовую адресную часть.

Большинство микропроцессоров имеет команды различной длины. Как правило, необходимы команды длиной в одно, два или три слова. Число битов, образующих двоичный код команды, не может быть произвольным, например 7, 12 или 14. Оно должно быть кратно длине байта (машинного слова). Следовательно, 8-разрядный микропроцессор может иметь команды длиной 8, 16 или 24 бит.

Длина команды зависит от длины используемого в ней адреса. Хотя различные микропроцессоры и располагают наборами команд, принципы адресации в них, как правило, одни и те же. Тип обращения (адресации) к данным принято называть *способом адресации*. В следующих разделах данной главы описываются различные способы адресации 8-разрядного микропроцессора, рассматриваемого в книге.

6.4. Неявная адресация

Однобайтовая команда 8-разрядного микропроцессора – это одна из 256 различных комбинаций 8 бит, образующих машинное слово (байт). Такого количества различных команд достаточно для рассматриваемого нами микропроцессора. Однако он располагает 65 536 областями памяти, для адресации которых адресная часть команды должны быть больше той, которую может предоставить 1-байтовая команда. Неприемлемым в данном случае является и такое решение, когда на месте адресной части размещаются сами данные. Однобайтовые данные нельзя разместить в 1-байтовой команде, часть битов которой должна всегда быть предоставлена коду операции.

Каким же образом можно использовать 1-байтовую команду для адресации к данным? Ответ таков: 1-байтовые команды не адресуются к данным, расположенным в памяти; они оперируют данными, загруженными в регистр, регистровую пару или данными, хранимыми в области памяти, адрес которой находится в регистровой паре. Например, 1-байтовая команда пересылки данных из регистра A в регистр B состоит из кода операции, адреса источника данных (регистра A) и адреса приемника данных (регистра B) (рис. 6.1). Адреса источника и приемника указаны в команде неявно; иногда говорят, что они «встроены» в команду. Вот почему такая адресация называется *неявной*.

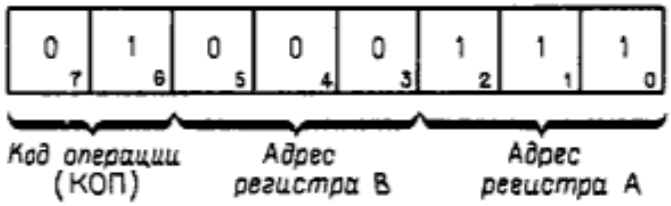


Рис. 6.1. Команда пересылки содержимого регистра A в регистр B, в которой использована неявная адресация. Код операции равен 01, адрес регистра A = 111, адрес регистра B = 000.

Однобайтовые команды выполняются быстрее любых других команд. Известно, что для выполнения какой-либо команды микропроцессору необходимо осуществить две последовательности операций, называемые циклом выборки и выполнения. В случае 1-байтовой команды микропроцессор затрачивает на это два микроцикла: один – на операцию выборки, другой – на операцию выполнения. В дальнейшем вы сможете убедиться, что для команд с другими способами адресации требуется большее количество микроциклов. Иначе говоря, команды с неявной адресацией отличаются наибольшей величиной быстродействия.

6.5. Непосредственная адресация

Этот способ адресации нетруден для понимания. Код операции команды с *непосредственной адресацией* размещается в первом байте. Сразу же за кодом операции следуют данные, занимающие 1 или 2 байт. Эти данные берутся не из памяти, их предоставляет машине программист при записи команды. Следовательно, при использовании данного способа адресации не требуется указание адреса памяти, необходим только код операции, после которого записываются данные.

Чтобы ответить на вопрос о том, в каких случаях целесообразно использование непосредственной адресации, рассмотрим пример программы загрузки аккумулятора 8-разрядным двоичным числом. Такая загрузка осуществляется при каждом выполнении программы. Указанную программу можно реализовать посредством команды, код операции которой «приказывает» микропроцессору загрузить в аккумулятор данные длиной в 1 байт, следующие непосредственно за кодом операции (рис. 6.2). Подобную команду называют ЗАГРУЗКА РЕГИСТРА НЕПОСРЕДСТВЕННАЯ. Конечно, в этом случае и код операции, и данные содержат информацию о двоичном коде.

Непосредственная адресация осуществляется микропроцессором за два микроцикла: в течение первого цикла производится

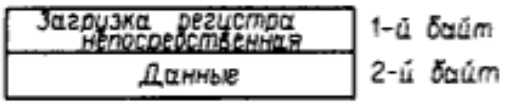


Рис. 6.2. Пример команды с непосредственной адресацией.

выборка команды, в течение второго – ее выполнение. Операции, задаваемые первым байтом команды (кодом операции), микропроцессор выполняет над данными, представленными ее вторым байтом. При использовании непосредственной адресации в команде предполагается размещение данных там же, где находится сама команда. Однако иногда возникает необходимость размещать данные в каком-либо другом месте памяти и иметь возможность адресоваться к ним. В таких случаях используется прямая адресация, описываемая в следующем разделе.

6.6. Прямая адресация

Команды с *прямой адресацией* могут иметь длину, равную 2 или 3 байт. Первый байт предназначен для кода операции, второй и, если имеется, третий – для адреса. Адрес указывает область памяти, в которой находятся подлежащие обработке данные. Совместное использование второго и третьего байтов команды позволяет адресоваться к любой из 65 536 областей памяти.

Прямая адресация, как правило, представляется наиболее естественной и простой для понимания. При неявной адресации адрес местоположения данных «встроен» в команду, и программист лишен возможности самостоятельно обращаться к данным по их адресу. При неспосредственной адресации данные указываются в самой команде, следуя сразу за кодом операции. В этом случае программист тоже не может адресоваться к данным. И только при прямой адресации у него имеется такая возможность, явным образом задавая адрес необходимых данных. Примером использования такой адресации может служить команда записи содержимого аккумулятора в память по адресу $000E_{16}$ (рис. 6.3). Этот адрес, заданный в шестнадцатеричной форме, занимает второй и третий байты 3-байтовой команды. (Адреса задаются в шестнадцатеричной форме, а содержимое адресуемых областей памяти – в двоичной.) Рис. 6.4 иллюстрирует, что происходит при выполнении этой команды. Перед выполнением команды (рис. 6.4, *а*) содержимое аккумулятора равно 1010 1010. В области памяти с адресом 000E содержится двоичное число 0000 0000. В области памяти с адресом 0003 находится код операции команды записи содержимого аккумулятора в память, а области с адресами 0004 и 0005 содержат информацию о местоположении необходимых данных, т. е. адрес 000E. После выполнения команды (рис. 6.4, *б*) копия содержимого аккумулятора оказывается размещенной в области памяти с адресом 000E.

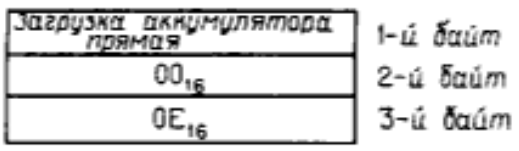


Рис. 6.3. Пример команды с прямой адресацией.

Рассмотрим еще один пример, поясняемый блок-схемой алгоритма (рис. 6.5) и описанием состояния памяти и аккумулятора по мере выполнения каждой операции (рис. 6.6). Перед началом выполнения рассматриваемой программы аккумулятор за-

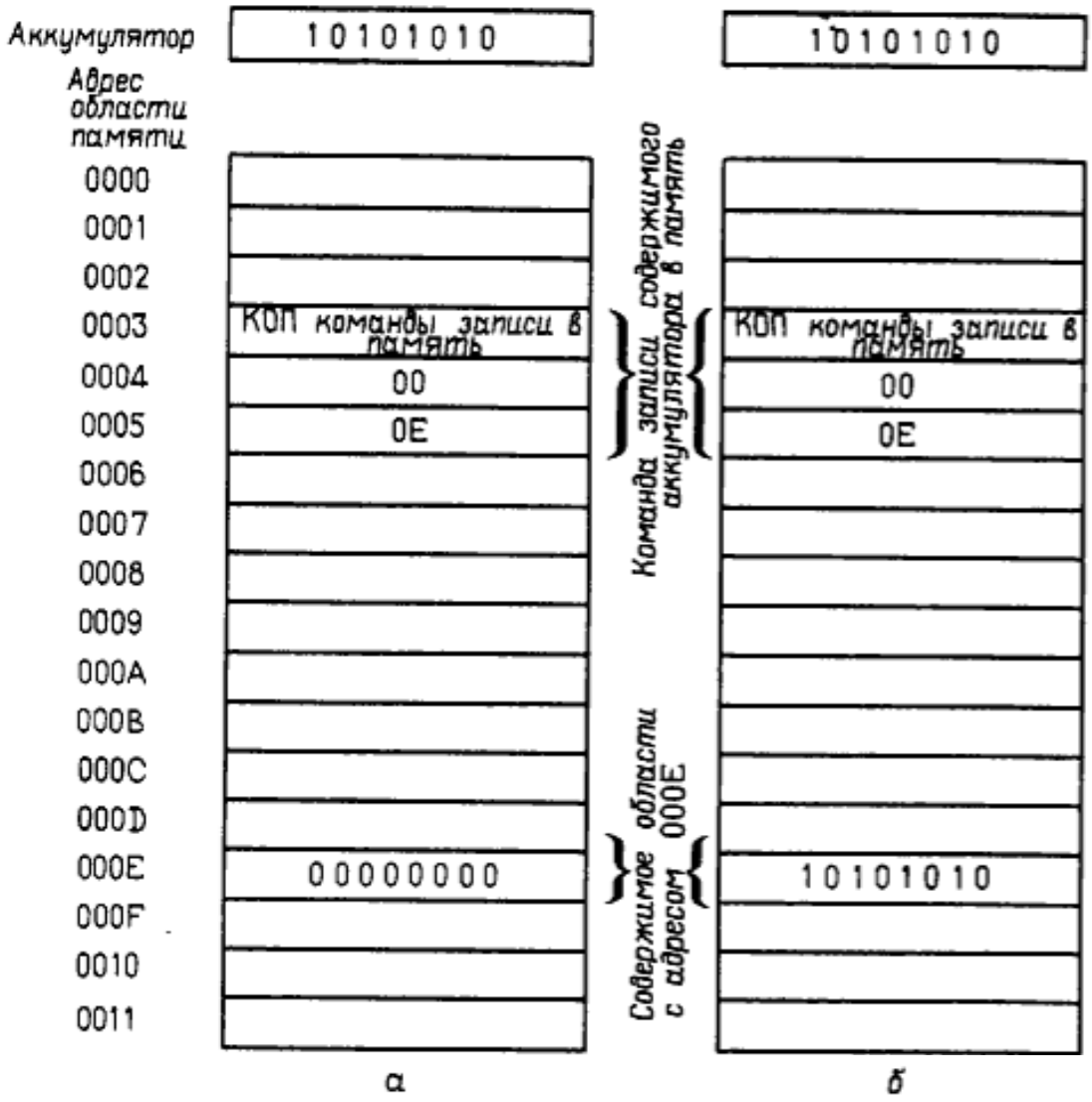


Рис. 6.4. Содержимое памяти и аккумулятора *а*) до и *б*) после выполнения команды записи содержимого аккумулятора в память (использование прямой адресации).

Рис. 6.5. Блок-схема алгоритма очистки аккумулятора с его последующей загрузкой содержимым области памяти с адресом 000E.

гружен результатом выполнения предыдущей операции. Первая команда данной программы – очистка аккумулятора – расположена в области памяти с адресом 0000. Занимающая три следующих байта памяти вторая команда – это команда загрузки в аккумулятор содержимого области памяти с адресом 000E, т.е. двоичного числа 0101 0101. Последняя команда – останов – занимает 1 байт памяти.

Содержимому аккумулятора можно было бы присвоить значение 0101 0101 также и посредством команды загрузки с непосредственной адресацией, но в этом случае загружаемые данные должны находиться в областях памяти, смежных с кодом операции этой команды. И если в дальнейшем эти же данные потребовались бы для другой команды, к ним трудно было бы получить доступ. При прямой адресации таких проблем не возникает, так как микропроцессор имеет возможность производить обращение к любой области памяти. В последнем примере благодаря использованию команды с прямой адресацией любое количество команд

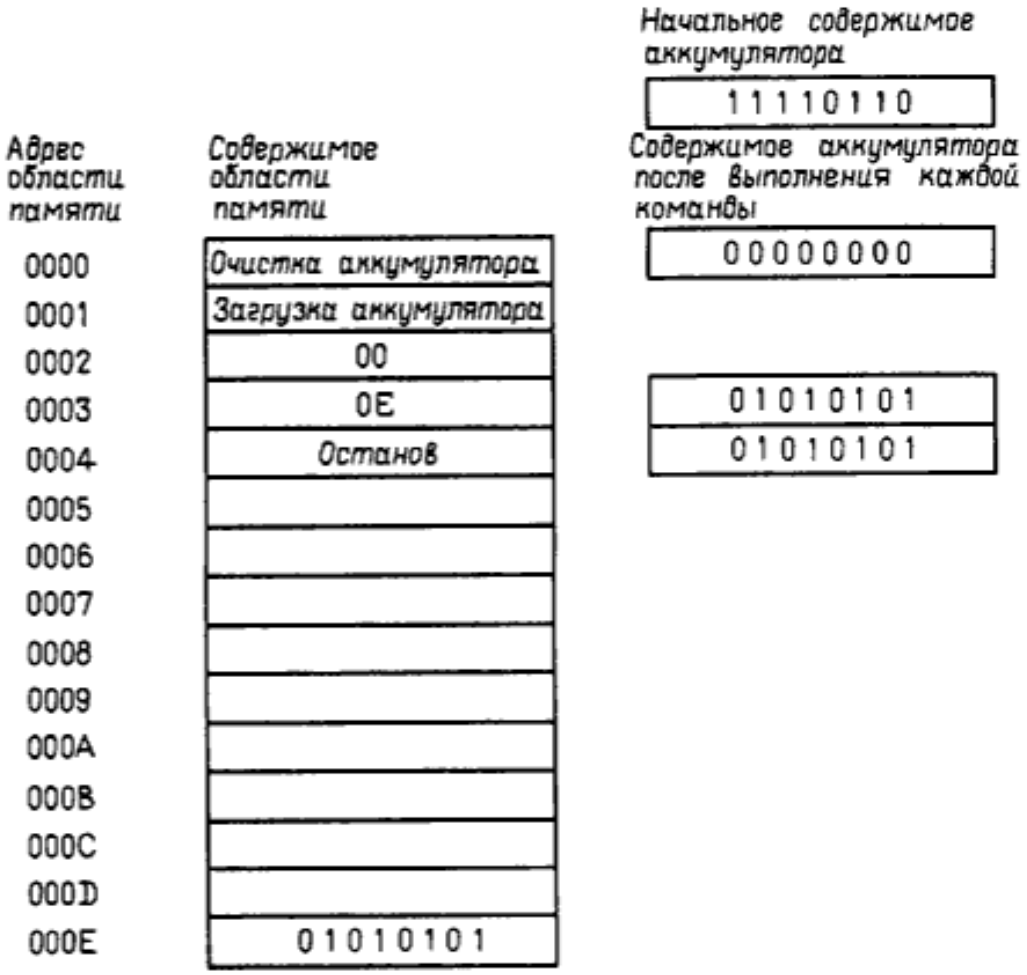


Рис. 6.6. Содержимое памяти и аккумулятора до и после выполнения программы, описываемой алгоритмом, блок-схема которого приведена на рис. 6.5.

Рис. 6.7. Блок-схема алгоритма очистки аккумулятора с последующей загрузкой его и регистра B содержимым области 000E.

программы может обращаться к области памяти с адресом 000E. Проиллюстрируем это, расширив рассмотренную программу добавлением одной команды (рис. 6.7). Программа по-прежнему размещается в памяти, начиная с адреса 0000. Но после очистки аккумулятора и его загрузки содержимым области памяти с адресом 000E программа не завершается, а имеет место загрузка регистра B содержимым той же области памяти. Содержимое памяти, аккумулятора и регистра B до и после выполнения команд программы показано на рис. 6.8. Начальное содержимое аккумулятора и регистра B определяется предшествующими операциями и может быть произвольным. При выполнении данной программы две различные команды с прямой адресацией пользуются одними и теми же данными, хранимыми в области памяти с адресом 000E. Отметим, что на содержимое регистра B не оказывают влияния ни очистка аккумулятора, ни его загрузка

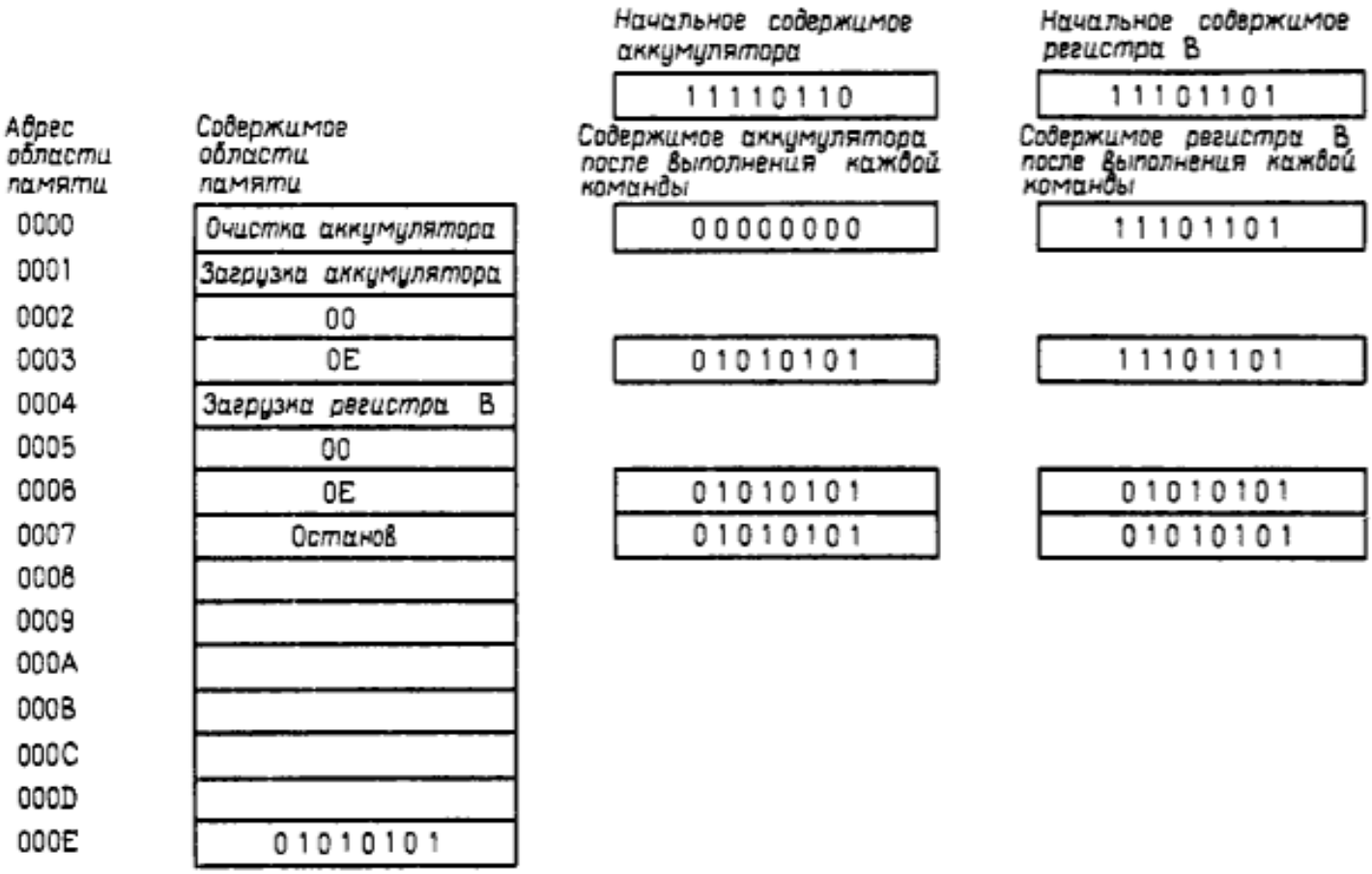


Рис. 6.8. Содержимое памяти, аккумулятора и регистра B до и после выполнения программы, описываемой алгоритмом, блок-схема которого приведена на рис. 6.7.

двоичным числом 0101 0101. Команда загрузки регистра B помещает в последний копию тех же данных, которые загружены в аккумулятор.

Применение прямой адресации связано с использованием дополнительного (по сравнению с ранее рассмотренными типами адресации) числа микроциклов. Во-первых, микропроцессору необходимо произвести выборку кода операции команды. После его декодирования следует извлечь из памяти еще 2 байт, являющихся значением адреса местоположения обрабатываемых данных. При каждом дополнительном обращении к памяти требуется еще один микроцикл. После извлечения кода операции и байтов адреса следует этап выполнения команды, на который затрачивается четвертый микроцикл. Следовательно, время выполнения команд с прямой адресацией в два раза больше, чем команд с непосредственной адресацией. Прямую адресацию целесообразно использовать, если необходимо иметь возможность размещать данные в любой области памяти. Поэтому при написании программы рекомендуется использовать как можно меньше команд с прямой адресацией, так как для их выполнения требуется больше времени.

В некоторых микропроцессорах используется прямая адресация, реализуемая за три микроцикла. В таких случаях команда имеет длину 2 байт: один – для кода операции, другой – для адреса. Но 1-байтовый адрес не позволяет адресоваться ко всем областям памяти объемом 65К. Например, в 8-разрядном микропроцессоре 8-битовое слово – второй байт команды с прямой адресацией – позволяет обращаться только к одной из 256 областей. Специальный код операции информирует, что такая команда имеет 1-байтовый адрес. Обычно отсчет этих адресов в памяти начинается с области, адрес которой равен 0. Диапазон значений, охватываемый такими 1-байтовыми адресами, принято называть *нулевой страницей* памяти. Так, адреса первых 256 областей памяти 8-разрядного микропроцессора принадлежат диапазону десятичных значений от 0 до 255 (двоичных – от 0000 0000 до 1111 1111, шестнадцатеричных – от 00 до FF). При адресации 2-байтовым словом

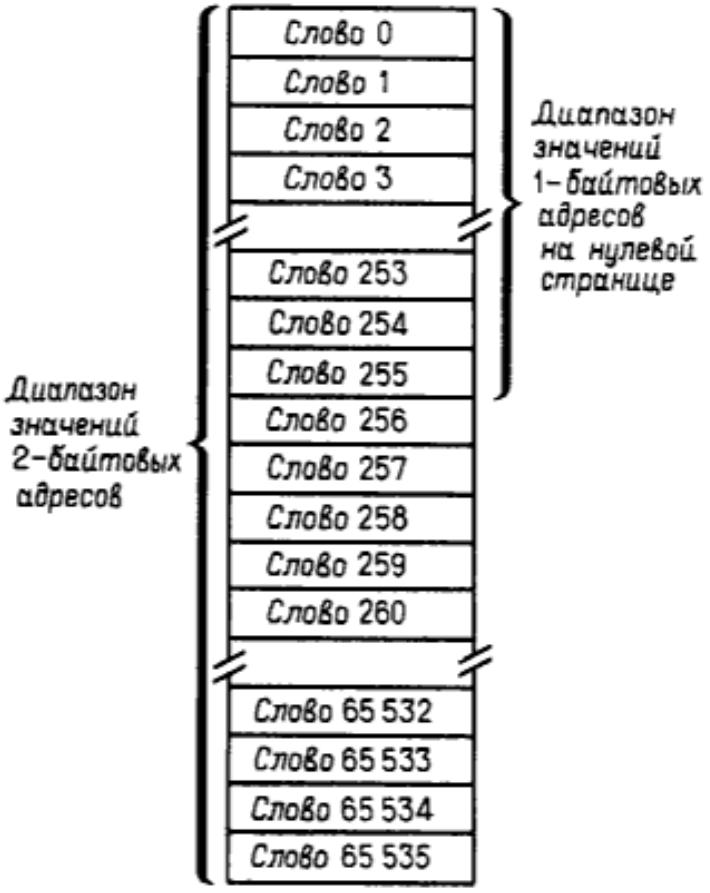


Рис. 6.9. Области памяти, адресуемые 1- и 2-байтовыми командами прямой адресации.

диапазон подобных десятичных значений простирается от 0 до 65 535 (двоичных – от 0000 0000 0000 0000 до 1111 1111 1111 1111, шестнадцатеричных – от 0000 до FFFF). На рис. 6.9 схематически изображены два описанных диапазона адресов – 1- и 2-байтовый.

Преимущество команд с 1-байтовыми адресами по сравнению с 2-байтовыми заключается в меньших затратах времени на их выполнение, поскольку при этом «экономится» один микроцикл. Команды с 1-байтовыми адресами целесообразно применять в тех случаях, когда требуется многократно обращаться к сравнительно небольшому количеству байтов данных, размещающихся в первых 256 областях памяти.

В 16-разрядном микропроцессоре с памятью объемом 65К прямая адресация предполагает использование только одного 16-битового адресного слова, так как оно позволяет адресоваться к любой области указанной памяти. Набор команд 16-разрядного микропроцессора состоит из команд длиной в одно или два слова.

Поскольку в 8-разрядных микропроцессорах применяются команды с прямой адресацией, имеющие длину в два или три слова памяти, возникает проблема их идентификации. Обычно такие команды различной длины имеют разные мнемонические имена, т. е. разные мнемонические обозначения кодов операций. Так 2-байтовые команды с прямой адресацией называют иногда *командами прямой адресации* или *командами адресации нулевой страницы*, а 3-байтовые – *командами расширенной прямой адресации*.

6.7. Косвенная адресация

Большинство микропроцессоров располагает еще одним способом адресации к памяти, реализуемой командами длиной только в одно слово. Такая адресация называется *косвенной* или иногда *косвенной регистровой*. Помимо кода операции в такой команде указывается номер регистра, содержимое которого – адрес местоположения данных в памяти. Этим подобные команды отличаются от команд непосредственной адресации, содержащие данные в самих себе, или от команд прямой адресации, которые включают адреса этих данных. Так, при использовании косвенной адресации в 8-разрядном микропроцессоре соответствующая команда указывает, в какой регистровой паре размещается адрес местоположения данных в памяти.

Косвенная адресация удобна при обращении к часто используемым областям памяти, и особенно в тех случаях, когда данные организованы в виде некоторого списка или *файла* (набора). Иначе говоря, использование косвенной адресации дает наибольший эффект при записи и чтении следующих друг за другом областей памяти. Примером может служить задача организации файла данных в памяти согласно блок-схеме алгоритма на рис. 6.10. В основу организации этого файла положен принцип косвенной регистровой адресации. Из порта ввода-вывода данные загружаются в аккумулятор, а затем записываются в следующие друг за другом области памяти. Используемые для этого команды подробно рассматриваются в гл. 10. Каждому пронумерованному блоку блок-схемы алгоритма соответствует одна команда. Процедура, описываемая блок-

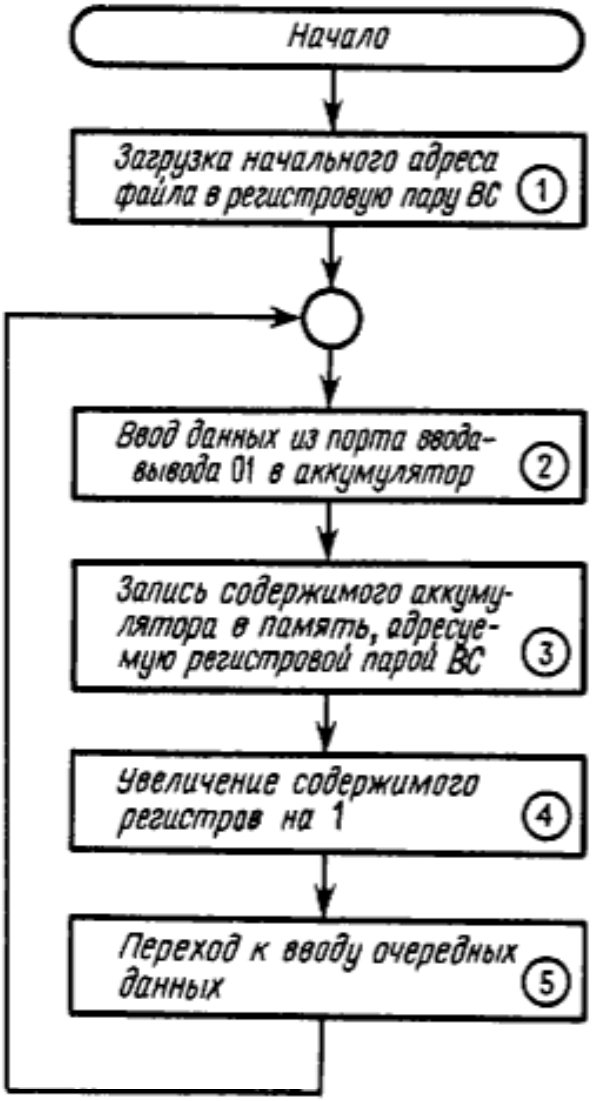


Рис. 6.10. Блок-схема алгоритма загрузки данных в смежные области памяти с использованием косвенной регистровой адресации.

схемой алгоритма, состоит из следующих шагов:

1. Задание начального адреса формируемого файла данных. Осуществляется посредством команды загрузки регистра, использующей непосредственную адресацию.

2. Загрузка аккумулятора данными из порта ввода-вывода с номером 01. Выполняется соответствующей командой ввода данных.

3. Запись копии содержимого аккумулятора в область памяти, адресуемую регистровой парой BC. Осуществляется командой записи в память, использующей косвенную адресацию. Обратите внимание, что в регистре, указываемом этой командой, находятся не загружаемые в память данные, а только их адрес. После выполнения загрузки необходимо принять новый байт данных и записать его в следующую, смежную область памяти.

4. Увеличение содержимого регистровой пары BC на 1. Выполнение этой операции необходимо для того, чтобы в регистровой паре оказался адрес области памяти, заполняемой следующей.

5. Переход к повторению загрузки аккумулятора очередными данными из порта ввода-вывода (возврат к блоку 2). Осуществляется после того, как регистр указывает адрес области памяти, «ожидающей» очередные данные. Содержимое регистра увеличивается каждый раз на 1. Если бы регистр перезагружался, механизм единичных приращений был бы нарушен и новые данные записывались в одну и ту же область.

Задания для самопроверки

10. Для какого из перечисленных здесь способов адресации памяти требуется наименьшее время для выполнения: а) прямой, б) прямой расширенной, в) непосредственной или г) неявной?

11. Что представляет собой второй байт команды с непосредственной адресацией: а) адрес области памяти, принадлежащей диапазону от 0_{10} до 255_{10}, б) 8-битовые данные, в) байт, легко доступный многим командам, или г) все перечисленное вместе?

12. Шестнадцатиразрядный микропроцессор использует 22-битовое адресное поле, обеспечивающее доступ к 4 194 304 областям памяти. Команда прямой адресации позволяет обращаться к любой области памяти, при этом ее первое слово занято кодом операции. Сколько байтов используется при формировании самой длинной подобной команды? Почему?

13. Косвенная адресация называется регистровой потому, что регистр, который указывает команда: а) содержит обрабатываемые данные, б) проверяет косвенным образом эти данные, в) содержит адрес данных или г) используется для областей памяти с адресами от 0_{10} до 255_{10}. Какое из этих утверждений справедливо?

14. Перечислите способы адресации памяти в порядке убывания скорости их выполнения.

15. Объясните, почему прямая (или косвенная) регистровая адресация является простейшей формой адресации памяти.

6.8. Простая микро-ЭВМ

Последующее изложение посвящено изучению того, как заставить микропроцессор выполнять необходимую нам работу. Иначе говоря, ставится задача анализа процесса решения задач на микро-ЭВМ, составной частью которой является микропроцессор. Вычислительные системы на базе микропроцессоров бывают самого различного типа. Примером небольшой микропроцессорной системы может служить построенная с использованием микропроцессора любая электронная игра. Вычислительная система на базе микропроцессоров с полным набором аппаратных средств может включать в свой состав устройство памяти, видеотерминал для ввода-вывода информации, устройство печати для копирования записей с экрана видеотерминала, накопители на гибких дисках в качестве внешней памяти. Подобная система может легко разместиться в корпусе небольшого видеотерминала (размером с переносной телевизор, диагональ экрана равна ~ 38 см), содержащего клавиатуру пишущей машинки. Такая система могла бы оказаться сравнительно мощной и небольшой по габаритам ЭВМ общего назначения на базе микропроцессора. В настоящее время подобные ЭВМ можно встретить и на производстве, и в личном владении пользователей.

Таким образом, микропроцессорная вычислительная система может иметь самую разнообразную конфигурацию аппаратных средств. Но в любом случае микропроцессор может функционировать только совместно с другими составными частями системы. А поэтому трудно изучать микропроцессор в отрыве от этих частей. Вот почему прежде всего следует обратить внимание на структурную схему микропроцессорной системы. Применительно к выбранному нами гипотетическому варианту эта схема приобретает вид в соответствии с рис. 6.11. Верхняя половина схемы – это описанная в гл. 3 структурная схема микропроцессора, нижняя половина схемы представляет собой память микро-ЭВМ и блок ввода-вывода. *Память* состоит только из шестнадцати 8-битовых слов, в то время как микропроцессор может адресоваться к 65 536 словам. Конечно, столь малая по объему память выбрана только для того, чтобы продемонстрировать работу микропроцессора. Используемый в последующих главах микропроцессор будет располагать бóльшим объемом памяти.

Память микропроцессора является запоминающим устройством произвольного доступа, т. е. произвести обращение к одному произвольно выбранному байту памяти столь же просто, как к любому другому ее байту. Это означает, что для доступа к определенному байту не нужно «проходить» через некоторую последовательность областей памяти. По команде данные могут быть записаны в любую область памяти или прочитаны из нее. Память обычно соединена с микропроцессором линиями трех видов: *адресными*, *данных* и *управления*.

Данные, поступающие по адресным линиям, используются для идентификации областей памяти или устройств, с которыми микропроцессору необходима связь. Адресные линии передают данные только из микропроцессора; в обратном направлении передача запрещена.

По линиям данных в любой определенный момент времени передается одно слово данных в микропроцессор или в обратном направлении. В отличие от адресных линий линии данных являются двунаправленными. По этим линиям передаются слова данных с внутренней шины микропроцессора в память или блок ввода-вывода либо из памяти или блока ввода-вывода на эту шину.

Линии управления позволяют микропроцессору управлять работой внешних устройств или контролировать работу последних. С помощью этих линий микропроцессор сообщает внешним устройствам, когда подавать данные на шину данных или когда получать их с шины. Благодаря сигналам, поступающим по линиям управления, передача данных происходит в должном порядке. Так, например, гарантируется, что распоряжение памяти «получить данные с шины» не поступит до момента установления стабильного значения адреса на выходах адресных линий.

Каждому слову памяти соответствует определенный номер – адрес его местоположения. Адрес используется всякий раз, когда необходимо записать слово данных в па-

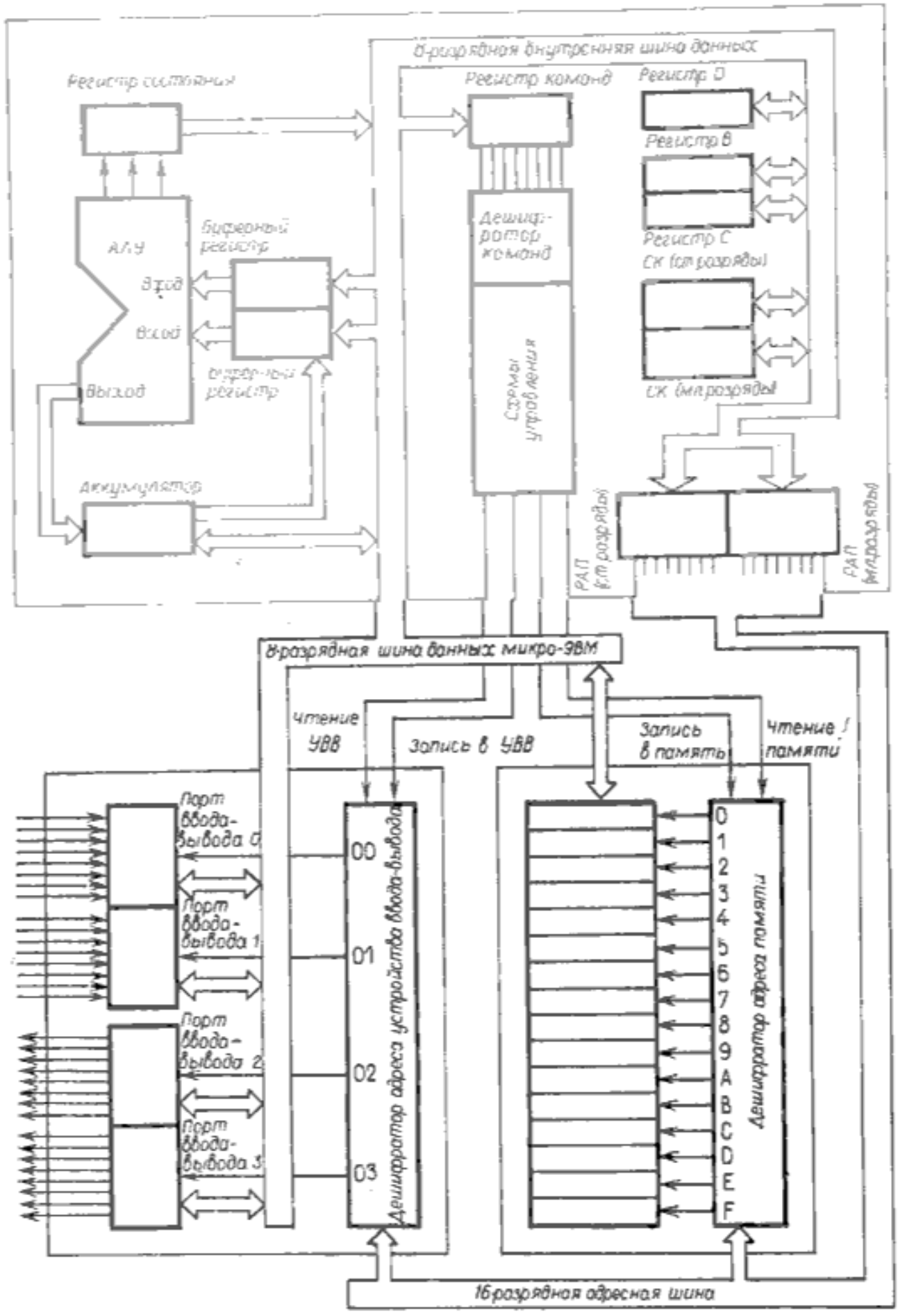


Рис. 6.11. Структурная схема 8-разрядной микро-ЭВМ с памятью объемом 16 слов и четырьмя портами ввода-вывода. (Адресные линии A0 – A3 предназначены для памяти, а линии A0 – A1 отведены для системы ввода-вывода. Все восемь линий шины данных (D0 – D7) используются устройством ввода-вывода и памятью.)

мять или прочесть слово из памяти. Адрес — двоичное число – появляется на адресной шине микропроцессора и декодируется *дешифратором адреса памяти*. Если на адресную шину помещен адрес допустимого значения, то схемы управления микропроцессора вырабатывают и посылают по линиям управления импульсные сигналы «Чтение памяти» или «Запись в память». Эти сигналы информируют микропроцессор о необходимости подачи данных на соответствующую шину или получения данных с шины для записи в выбранную область памяти. При записи данных в память предыдущее содержимое соответствующей ее области стирается и заменяется записываемой информацией. Чтение данных из памяти не меняет содержимое области, в которой они находятся. Одни и те же данные можно считывать любое количество раз. Принято говорить, что операция чтения является операцией, не разрушающей информацию.

Блок ввода-вывода микро-ЭВМ включает в себя *порты ввода-вывода* с номерами 00, 01, 02 и 03. Порты 00 и 01 являются входными. Каждый порт имеет восемь входных линий. При адресации порта 00 или 01 данные, поступающие по этим линиям, подаются на шину данных микро-ЭВМ. Затем микропроцессор помещает эти данные в аккумулятор.

Порты 02 и 03 являются выходными; каждый из них имеет восемь выходных линий. Состояние каждой выходной линии хранится в буфере данных. При обращении к порту 02 или 03 8 бит данных, имеющихся на шине данных микро-ЭВМ, записываются в буфер данных адресуемого выходного порта. Эти данные буфер хранит для возможного их использования устройствами, внешними по отношению к микро-ЭВМ. Данные сохраняются до следующего обращения к этому порту или до тех пор, пока не включат питание микро-ЭВМ.

Порты 00, 01, 02 и 03 называют *портами параллельного ввода-вывода*, поскольку данные подаются во «внешнюю среду» по восьми параллельным линиям. Принцип такого «параллелизма» заложен в основу организации системы ввода-вывода. При *последовательном вводе-выводе* соответствующий порт должен использовать регистр сдвига для преобразования параллельно вводимых (выводимых) данных в последовательные, и наоборот. Последовательный ввод-вывод применяется, когда возникает необходимость передавать данные на большие расстояния. В пункте приема устанавливается преобразователь последовательного кода в параллельный, подаваемый по восьми параллельным линиям. Порт последовательного ввода-вывода более подробно (в том числе применительно к рассматриваемой гипотетической микро-ЭВМ) описывается в гл. 12.

Дешифратор адреса устройства ввода-вывода подобен дешифратору адреса памяти. Поскольку имеются только четыре порта ввода-вывода, достаточно располагать двумя адресными линиями. Дешифратор адреса устройства ввода-вывода декодирует все адреса, посылаемые микропроцессором. Однако порты ввода-вывода «откликаются» на запрос, посылаемый в форме их адресов, только тогда, когда имеет место совпадение импульсов «Чтение ввода-вывода» или «Запись ввода-вывода» с декодированием этих адресов.

Шина микро-ЭВМ предназначена для передачи нескольких видов сигналов. Она состоит из восьми двунаправленных линий данных, т. е. по этим линиям можно передавать данные как из микропроцессора, так и к нему. Эту шину можно рассматривать как продолжение внутренней шины данных. 16-разрядная шина соединена с дешифраторами адреса, расположенными в памяти и блоке ввода-вывода. Шестнадцать адресных линий адреса используются для адресации как памяти, так и устройств ввода-вывода. Четыре дополнительные линии предназначены для передачи сигналов чтения и записи памяти и портов ввода-вывода. Кроме того, по адресной шине передаются сигналы питания, некоторые сигналы синхронизации и управления, а также с помощью этой шины осуществляется «заземление» всех блоков. Более подробно этот вопрос рассматривается в последующих главах.

Задания для самопроверки

16. Если микропроцессор имеет 16-разрядную адресную шину, то он может адресоваться: а) к 65 536 словам памяти, б) к 16 8-битовым словам памяти, в) к 65 536 8-би-

товым словам памяти или г) к 32 768 1-байтовым словам памяти. Какое из сделанных утверждений является верным?

17. Какого рода информация передается по линиям шины микро-ЭВМ: а) данные, б) адрес памяти, в) сигналы управления и питание или г) все перечисленные виды информации?

18. У микропроцессора имеются четыре порта ввода-вывода, каждый из которых содержит восемь линий данных. Как называются эти порты: а) порты параллельного ввода-вывода, б) порты вывода адресов, в) порты последовательного ввода или г) порты последовательного вывода?

19. Для чего предназначена 8-разрядная двунаправленная шина данных: а) соединять память и устройства ввода-вывода с внутренней шиной микропроцессора; б) адресоваться к устройствам ввода-вывода; в) перемещать данные из микропроцессора в память или г) пересылать входные данные на внутреннюю шину данных микропроцессора?

20. Какие из нижеперечисленных сигналов передаются по линиям управления микропроцессора: а) «Чтение памяти», б) «Запись в память», в) «Чтение-запись ввода-вывода» или г) все указанные сигналы?

21. Какие из перечисленных ниже характеристик справедливы по отношению к адресной шине микропроцессора: а) ширина шины равна 8 бит, б) ширина шины равна 16 бит, в) шина является двунаправленной или г) шина является однонаправленной?

Упражнения

6.1. Команда микропроцессора – это слово, имеющее единственное значение для воспринимающих его схем управления работой ЭВМ. Является это слово десятичным, шестнадцатеричным, двоичным или восьмеричным?

6.2. О чем из нижеперечисленного команда информирует микропроцессор: данные находятся в памяти, данные находятся в регистре, длина данных составляет 8 бит или о том, что делать?

6.3. Какой может быть длина команды 8-разрядного микропроцессора: 8 бит, 16 бит, 24 бит или любой из указанных здесь величин?

6.4. Что означает термин «набор команд»?

6.5. Из каких двух частей состоит команда? Каково назначение каждой части?

6.6. Что такое мнемоническое имя команды микропроцессора?

6.7. Зачем используется мнемоническое обозначение?

6.8. Какими из указанных здесь признаков могут отличаться команды, имеющие одинаковый код операций: длиной, способом адресации, словом данных или скоростью выполнения?

6.9. Объектом воздействия кода операции является его операнд. Почему некоторые операнды задаются в виде двоичных (шестнадцатеричных или восьмеричных) чисел, а другие – как аббревиатура английских слов?

6.10. Каким по порядку словом команды (из трех используемых слов) является ее код операции, если другие слова, образующие команду, это – данные или адреса их местоположения?

6.11. Какому числу байтов (от 1 до 8) равна длина команды с неявной адресацией 8-разрядного микропроцессора?

6.12. Некоторые 2-байтовые команды содержат данные во втором байте. Какой способ адресации при этом используется: прямая, непосредственная, неявная или косвенная регистровая?

6.13. Если второй и третий байты команды 8-разрядного микропроцессора являются данными, то какой способ адресации при этом используется: прямая, непосредственная, неявная или косвенная регистровая?

6.14. При использовании какого способа адресации (прямой, непосредственной, неявной или косвенной регистровой) для выполнения команды требуется наибольшее количество микроциклов?

6.15. Какой способ адресации из указанных здесь позволяет производить обращение к любой области памяти посредством команды длиной в одно слово: прямая, непосредственная, неявная или косвенная регистровая?

6.16. Попытайтесь догадаться, каково назначение перечисляемых ниже команд, и какой способ адресации в них используется:

CLA A	ADD 56	AND B	HLT
STA 34 DE	INC B	LDA 2E	JMP 105A.

6.17. С каким из перечисленных ниже узлов микро-ЭВМ соединена внутренняя шина данных микропроцессора: с адресной шиной, шиной данных, дешифратором адреса памяти или дешифратором адреса устройства ввода-вывода?

6.18. Какие из указанных ниже сигналов не передаются по линиям управления микропроцессора: «Чтение памяти», «Запись в память», адреса памяти или «Чтение-запись ввода-вывода»?

Ответы на вопросы заданий для самопроверки

1. а. 2. г. 3. б. 4. б.
5. 303_8 – это код операции, а JMP – мнемоническое обозначение, являющееся сокращением английского названия операции, выполняемой командой. Код операции – это слово, выраженное цифрами в некоторой системе счисления (например, восьмеричной).
6. CLA – Clear (Очистка), NOP – No Operation (Нет операции), MOV – Move (Пересылка), HLT – Halt (Останов), INC – Increment (Приращение положительное), ADD – Add (Сложение), AND – Logical AND (Логическое И), DEC – Decrement (Приращение отрицательное).
7. Нет. Мнемоническое обозначение – это только сокращенное название команды, не имеющее ничего общего с ее длиной как двоичного кода.
8. Код операции.
9. Да. Необходимо каким-либо образом отличать одну команду от другой.
10. г. 11. б.
12. 4 байт потому, что необходимо более 16 бит, а добавление третьего слова увеличивает общую длину на 16 бит (2 байт).
13. в.
14. Неявная, непосредственная, прямая.
15. Косвенная регистровая адресация предполагает предварительную загрузку адреса в регистровую пару, а в случае прямой адресации эти адреса содержатся во втором и третьем байтах команды.
16. а. 17. г. 18. а. 19. б. 20. г. 21. г.

Глава 7. Команды пересылки данных

В данной главе рассматриваются более подробно команды микропроцессора, служащие для пересылки данных, т. е. для передачи данных в микропроцессорной системе из одного места в другое. Это очень важная группа команд, так как при использовании микропроцессора постоянно происходит пересылка данных между различными устройствами и областями памяти.

Команды, описываемые в этой главе, входят в набор команд нашего гипотетического микропроцессора; при этом они аналогичны командам того же назначения, которые используются почти во всех серийно выпускаемых микропроцессорах, поэтому ознакомление с ними весьма полезно. Читатель ознакомится с элементами, общими для наборов команд многих микропроцессоров, а также научится анализировать сведения, обычно приводимые при описании наборов команд конкретных микропроцессоров. Эти знания существенны для разработки программ, проектирования аппаратных средств и работы с микропроцессорами.

После изучения набора команд микропроцессора подробно рассматривается пример программы, который наглядно показывает, каким образом микропроцессор решает задачи путем выполнения команд.

7.1. Советы по изучению набора команд

В следующих ниже разделах каждой новой команде отводится отдельный абзац. Такой абзац в сочетании с сопровождающей его графической информацией содержит все данные, необходимые для изучения команды.

В большинстве случаев названия команд микропроцессора в достаточной степени характеризуют их назначение. Если же это не так, то по ходу изложения приведены необходимые пояснения.

После ознакомления с описанием команды читатель должен усвоить следующие ее характеристики:

1. Название.
2. Используемые способы адресации.
3. Мнемоническое обозначение.
4. Размещение команды в памяти.
5. Представление действий, выполняемых командой, с помощью логических символов.
6. Объяснение действий, выполняемых командой.
7. Длину команды.
8. Воздействие результата выполнения команды на регистр состояния микропроцессора.

Все абзацы, содержащие описание команд, имеют одинаковую форму предста-

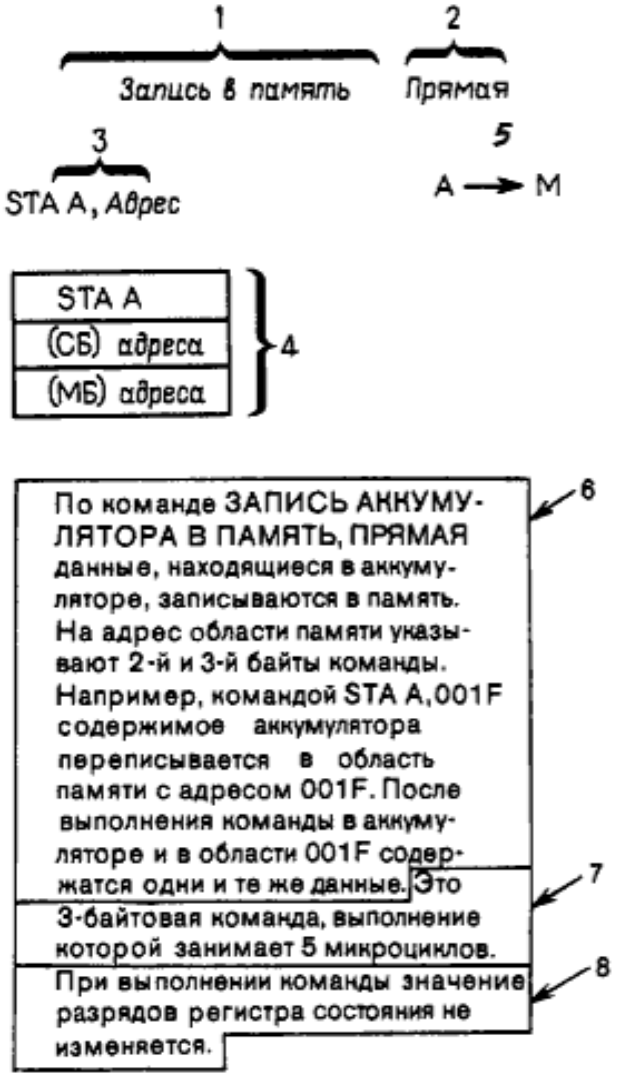


Рис. 7.1. Пример описания команды.

вления. Впоследствии вы убедитесь, что в аналогичной форме обычно представляют описание наборов команд изготовители микропроцессоров.

На рис. 7.1 показан пример описания одной из команд нашего гипотетического микропроцессора. Цифрами отмечены фрагменты информации, представляющие собой приведенные выше характеристики команды. Каждому из прямоугольников, изображенных на рисунке, соответствует 1 байт команды.

По мере ознакомления с описанием команд постоянно проверяйте, получили ли вы представление обо всех указанных здесь характеристиках каждой команды. Впоследствии приведенный выше список характеристик пригодится при анализе наборов команд серийных микропроцессоров. Если изготовитель микропроцессора не указал в описании всех восьми параметров каждой команды, значит, он не привел необходимых сведений, относящихся к микропроцессору.

При изучении наборов команд выясняется, что разные изготовители используют различные наименования для одних и тех же команд. Убедитесь, что вы точно установили функцию каждой команды микропроцессора, с которым имеете дело.

7.2. Команды пересылки данных

В этом разделе мы рассмотрим *команды пересылки данных*, называемые также *командами перемещения данных*. Они служат для пересылки данных в различные устройства хранения информации, которыми располагает микро-ЭВМ, а также для пересылки данных из этих устройств. К числу областей хранения информации относятся как ячейки памяти, так и регистры. В зависимости от того, какие устройства микро-ЭВМ участвуют в пересылке данных, различают команды: ЗАГРУЗКА, ПЕРЕСЫЛКА и ЗАПИСЬ В ПАМЯТЬ.

Команды пересылки данных, возможно, следовало бы называть командами «копирования», потому что практически они осуществляют перемещение именно копии данных. Так, например, одна из этих команд перемещает данные из аккумулятора в область памяти. После выполнения данной команды и в области памяти, и в аккумуляторе находятся одни и те же данные. Очень редко в ходе выполнения команды пересылки данных разрушаются данные, находящиеся в исходном месте.

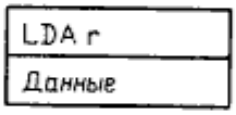
Как и любая другая команда, команда пересылки данных состоит из двух частей: кода операции и адресной части. Во всех командах пересылки данных должны быть указаны как источник, так и место назначения данных.

Код операции в команде пересылки данных задает источник данных и способ адресации. Адресация может быть одного из трех видов: непосредственная, прямая и косвенная.

Восьмиразрядный микропроцессор может иметь 1-, 2- и 3-байтовые команды пересылки данных. Ниже будут рассмотрены команды пересылки всех форматов.

ЗАГРУЗКА РЕГИСТРА НЕПОСРЕДСТВЕННАЯ

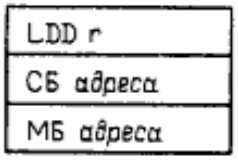
LDA r,Данные Данные → r



По команде ЗАГРУЗКА РЕГИСТРА НЕПОСРЕДСТВЕННАЯ в регистр r загружаются данные, находящиеся во втором байте команды. Например, при выполнении команды LDA A данные загружаются в аккумулятор. С помощью команды LDA B производится загрузка данных в регистр B. Команда имеет 2 байт и выполняется за три микроцикла. Если при ее выполнении результат оказывается равным 0 или содержит 1 в старшем разряде, то устанавливается в 1 соответствующий разряд регистра состояния[1].

ЗАГРУЗКА РЕГИСТРА ПРЯМАЯ

LDD r,Адрес M → r

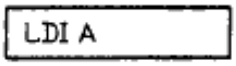


При выполнении команды ЗАГРУЗКА РЕГИСТРА ПРЯМАЯ данные переписываются из адресуемой области памяти в регистр r. Второй и третий байты команды указывают адрес области памяти, данные которой копируются. Так, например, по команде LDD C, 0F1C данные, находящиеся в области памяти 0F1C, переписываются в регистр C. Это 3-байтовая команда, выполнение которой занимает четыре микроцикла. Если результат операции равен 0 или содержит 1 в старшем разряде, то устанавливается в 1 соответствующий разряд регистра состояния.

[1] Напомним, что разряд отрицательного результата в регистре состояния нашего микропроцессора устанавливается в 1, если появляется 1 в старшем разряде результата.

ЗАГРУЗКА АККУМУЛЯТОРА КОСВЕННАЯ

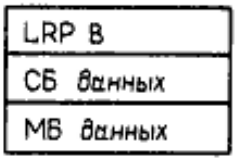
LDI A M → A



По команде ЗАГРУЗКА АКУУМУЛЯТОРА КОСВЕННАЯ данные переписываются из области памяти в аккумулятор. Содержимое регистровой пары BC указывает адрес области памяти – источника данных. Для того чтобы рассматриваемая команда могла быть использована, в регистровую пару BC необходимо предварительно занести нужный адрес. Это 1-байтовая команда, выполняющаяся за два микроцикла. Если результат выполнения команды равен 0 или содержит 1 в старшем разряде, то устанавливается в 1 соответствующий разряд регистра состояния.

ЗАГРУЗКА РЕГИСТРОВОЙ ПАРЫ НЕПОСРЕДСТВЕННАЯ

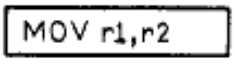
LRP B,Данные M → C
(M + 1) → B



С помощью команды ЗАГРУЗКА РЕГИСТРОВОЙ ПАРЫ НЕПОСРЕДСТВЕННАЯ в 16-разрядную пару регистров BC заносятся данные, содержащиеся во втором и третьем байтах команды. Это 3-байтовая команда, для выполнения которой необходимы три микроцикла. (По сравнению с последовательностью команд LDA B и LDA C использование данной команды позволяет сэкономить три микроцикла.) Значение разрядов регистра состояния при выполнении команды LRP B не изменяется.

ПЕРЕСЫЛКА ИЗ РЕГИСТРА В РЕГИСТР

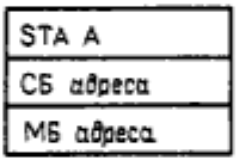
MOV r1,r2 r2 → r1



По команде ПЕРЕСЫЛКА ИЗ РЕГИСТРА В РЕГИСТР в регистр r1 загружается копия данных, содержащихся в регистре r2. Так, при выполнении команды MOV B,A копия данных, находящихся в аккумуляторе (регистре A), записывается в регистр B. Это 1-байтовая команда, реализуемая за два микроцикла. Если результат выполнения команды содержит 1 в старшем разряде или равен 0, то устанавливается в 1 соответствующий разряд регистра состояния.

ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ ПРЯМАЯ

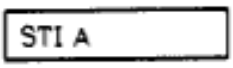
STA A,Адрес A → M



При использовании команды ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ ПРЯМАЯ данные из аккумулятора записываются в память. Второй и третий байты команды указывают на область памяти, в которую производится запись данных. Например, по команде STA, A,001F содержимое аккумулятора записывается в область памяти 001F. Это 3-байтовая команда, реализация которой занимает пять микроциклов. Значение разрядов регистра состояния в результате выполнения операции не изменяется.

ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ КОСВЕННАЯ

STI A A → M



При использовании команды ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ КОСВЕННАЯ данные переписываются из аккумулятора в память. Содержимое регистровой пары BC указывает адрес области памяти, в которую осуществляется запись данных. (Для того чтобы можно было воспользоваться данной командой, в пару регистров BC необходимо предварительно загрузить нужный адрес.) Это 1-байтовая команда, выполнение которой занимает два микроцикла. Значение разрядов регистра состояния при выполнении данной команды не изменяется.

Задания для самопроверки

1. Для каких из указанных ниже целей служит команда ПЕРЕСЫЛКА:
   а) для пересылки второго байта команды в указанный регистр;
   б) пересылки содержимого аккумулятора в память;
   в) пересылки содержимого одного регистра в другой регистр;
   г) пересылки косвенно адресованных данных в регистр?
2. Объясните различие между командами LDA A и LDD A. Какая из них выполняется дольше?
3. Покажите, как можно использовать команды LDA B и LDA C вместо команды LRP B. В чем недостаток метода, при котором вместо одной команды применяются две?
4. Объясните различие между командами STA A и LDD A. Почему они не выполняют одних и тех же действий?
5. В регистре B находятся какие-то данные. Необходимо записать их в область памяти с адресом 01FF Какие две команды можно применить для этого? Приведите соответствующие блок-схемы алгоритмов и кратко опишите действия, выполняемые каждой из этих команд.
6. Регистровая пара BC используется в качестве 16-разрядного регистра. С интервалами времени ~0,1 с вступает в работу программа-счетчик. В начале ее выполнения в регистровую пару BC загружаются данные, к ним прибавляется необходимое количество единиц, после чего результирующее 16-разрядное число вновь записывается в память. Это 16-разрядное число размещается в областях памяти 00F0 (старший байт) и 00F1 (младший байт). Какие команды пересылки данных необходимы для выполнения этих действий? В каких областях памяти размещаются эти команды? Закончите разработку блок-схемы, приведенной на рис. 7.2.

Рис 7.2. Блок-схема программы к п. 6 заданий для самопроверки.

7.3. Пример программы

В этом разделе на примере программы показано, каким образом микропроцессор шаг за шагом осуществляет решение задачи. Воспользуемся рассмотренной выше программой, служащей для приема данных из порта ввода-вывода 01 и записи данных в смежные области (рис. 6.10). Другими словами, эта программа создает файл данных.

Данные поступают в порт 01. По команде ВВОД данные передаются из порта 01 в аккумулятор. Из аккумулятора они пересылаются в последовательные области памяти. Создание файла данных начинается с области памяти с адресом 000E.

Внимательно ознакомившись с блок-схемой, можно увидеть, что по существу она представляет лишь часть программы решения задачи – в ней отсутствует завершающая часть. Впоследствии мы узнаем, каким образом можно закончить выполнение программы.

В данной программе использованы три команды, которые пока не были изучены, – ВВОД, ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ и ПЕРЕХОД. В дальнейшем указанные команды будут подробно рассмотрены. Для того чтобы разобраться в данном примере, достаточно иметь о них лишь общее представление.

Команду ВВОД можно рассматривать как 2-байтовую команду ЗАГРУЗКА АККУМУЛЯТОРА ПРЯМАЯ.

При каждом выполнении команды ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ к содержимому регистров ВС прибавляется 1.

По команде ПЕРЕХОД в счетчик команд загружается новое значение, содержащееся во втором и третьем байтах этой команды.

Знакомясь с последовательностью выполнения программы, мы рассмотрим разные состояния процессов, протекающих в микропроцессорной системе. Эти состояния будут соответствовать моментам завершения каждого микроцикла. Таким образом, мы рассмотрим состояния системы после каждого цикла выборки и каждого цикла выполнения. Мы увидим, как поступают данные из порта ввода-вывода в регистр, а затем в область памяти.

В процессе ознакомления с примером обязательно обращайте внимание на изменения в состоянии блоков микропроцессорной системы, которые выделены черным цветом. Сравнивайте каждый из рисунков с предшествующим. Проследите все изменения состояния системы, имеющие место в ходе выполнения программы.

Содержимое всех областей памяти и регистров указано в шестнадцатеричной системе счисления. При этом индексы обозначения системы счисления не приведены. Необходимо ясно понимать, что реальное содержимое области памяти представляет собой набор из 8 бит, каждый из которых имеет значение логической 1 или логического 0. Таким образом, не следует забывать, что вы имеете дело с двоичными устройствами, а шестнадцатеричные числа являются лишь удобным средством отображения состояния этих устройств.

На некоторых из приведенных ниже схем показано содержимое не всех областей памяти и регистров, т. е. отсутствуют находящиеся в них данные. Это означает, что данные, содержащиеся в таких регистрах и областях памяти, не представляют интереса для изучения рассматриваемых здесь вопросов. Конечно, в каждом регистре и во всех областях памяти в любой момент времени имеются какие-то данные, значения которых лежат в диапазоне от всех логических 0 (00) до всех логических 1 (FF).

На рис. 7.3 показано состояние системы, предшествующее ее готовности к выполнению программы. Программа хранится

в областях с адресами 0000–0009. Выполнение программы начинается с области 0000. Поэтому необходимо присвоить счетчику команд (СК) значение, при котором все его разряды установлены в 0. Выполнение программы начинается с команды, которая была ранее загружена в область памяти с адресом 0000.

На данном рисунке содержимое других регистров и областей памяти не показано. Как отмечено выше, это означает, что их содержимое в момент времени, которому соответствует рисунок, несущественно.

На рис. 7.4 показано состояние системы после выполнения первого цикла выборки. Реализация команды ЗАГРУЗКА РЕГИСТРОВОЙ ПАРЫ НЕПОСРЕДСТВЕННАЯ включает один цикл выборки и два цикла выполнения, т.е. всего три микроцикла. Как показано на рис. 7.4, длина команды составляет 3 байт. Данная 3-байтовая команда хранится в областях памяти 0000, 0001 и 0002.

Указанный рисунок иллюстрирует выполнение цикла выборки первой команды. Содержимое счетчика команд загружается в регистр адреса памяти. При этом на адресную шину поступает двоичный адрес области памяти 0000. Как только значения сигналов на адресной шине приобретают достоверные, т.е. установившиеся, значения, устройство (схемы) управления выдает сигнал «Чтение памяти». При этом декодируется адресная информация, поступающая по адресной шине, в результате чего оказывается выбранной область памяти 0000.

При получении сигнала «Чтение» содержимое области 0000 (команда ЗАГРУЗКА РЕГИСТРОВОЙ ПАРЫ НЕПОСРЕДСТВЕННАЯ) выводится на шину данных микро-ЭВМ.

Так как имеет место цикл выборки, команда передается в регистр команд микропроцессора. Во время цикла выборки регистр команд подключен к внутренней шине данных микропроцессора[1].

Цикл выборки первой команды завершен. Первая команда программы передана в регистр команд микропроцессора. Теперь микропроцессор может начать выполнение этой команды.

На рис. 7.5 показано, что происходит при реализации первого цикла фазы выполнения команды ЗАГРУЗКА РЕГИСТРОВОЙ ПАРЫ НЕПОСРЕДСТВЕННАЯ, т.е. представлены результаты реализации второго из трех микроциклов этой команды. При декодировании команды ЗАГРУЗКА РЕГИСТРОВОЙ ПАРЫ НЕПОСРЕДСТВЕННАЯ устройство управления сразу обнаруживает, что это 3-байтовая команда. Положительное приращение содержимого счетчика команд осуществляется три раза. Теперь счетчик указывает на следующую команду. Эта команда находится в области памяти с адресом 0003.

Регистр адреса памяти указывает на область 0001, так как устройству управления известно, что второй байт команды находится в этой области. В области 0001 содержатся данные, которые необходимо поместить в старший байт регистровой пары ВС.

Адрес памяти 0001 подается на адресную шину и декодируется дешифратором адреса памяти. Дешифратор указывает на область памяти 0001. Затем содержимое области 0001 (число 00) подается на шину данных. Данные (число 00) загружаются в старший байт регистровой пары общего назначения ВС. На этом заканчивается первая половина фазы выполнения команды.

Вторая половина фазы выполнения первой команды представлена на рис. 7.6. (Счетчик команд по-прежнему указывает на область с адресом 0003, т.е. на адрес следующей команды. Изменять содержимое счетчика команд пока еще рано.)

Устройство управления принуждает сейчас регистр адреса памяти указывать на область памяти 0002, в которой содержится третий байт команды ЗАГРУЗКА РЕГИСТРОВОЙ ПАРЫ НЕПОСРЕДСТВЕННАЯ. Из регистра адреса памяти этот адрес подается на адресную шину, и дешифратор адреса памяти его декодирует. Устройство управления вырабатывает сигнал «Чтение памяти», и содержимое области памяти 0002 поступает на шину данных. Данные (число 0E) из этой области передаются в младший байт регистровой пары ВС.

[1] Напомним, что в ходе цикла выборки производится загрузка команды в регистр команд. Все остальные циклы являются циклами выполнения, хотя в процессе их реализации и может происходить выборка данных из памяти. Они называются так потому, что их реализация составляет собственно процесс выполнения команды.

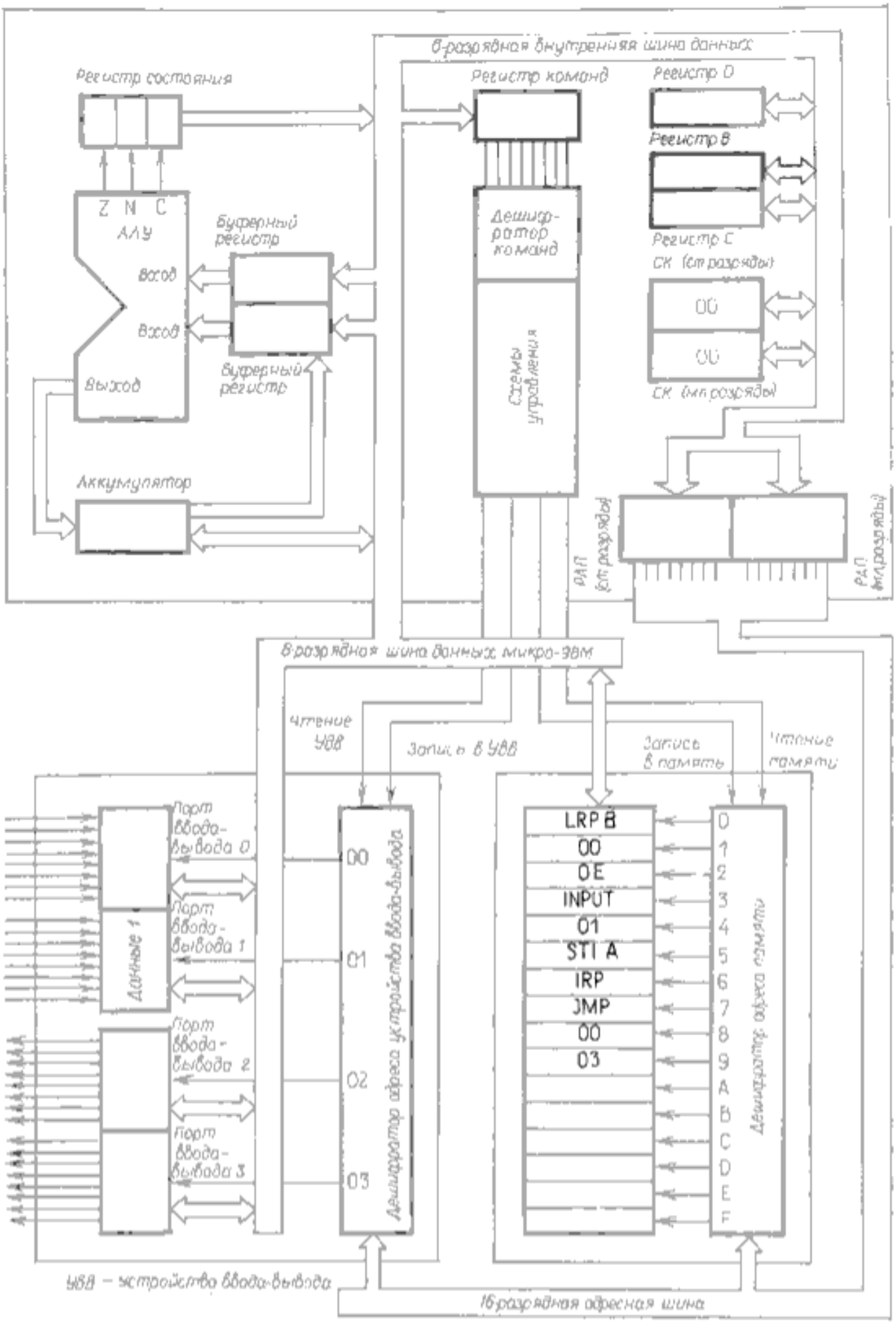


Рис. 7.3. Микро-ЭВМ ожидает начала выполнения программы. (На выходах дешифратора адреса памяти номера областей с 0000 до 000F указаны как соответственно номера от 0 до F.)

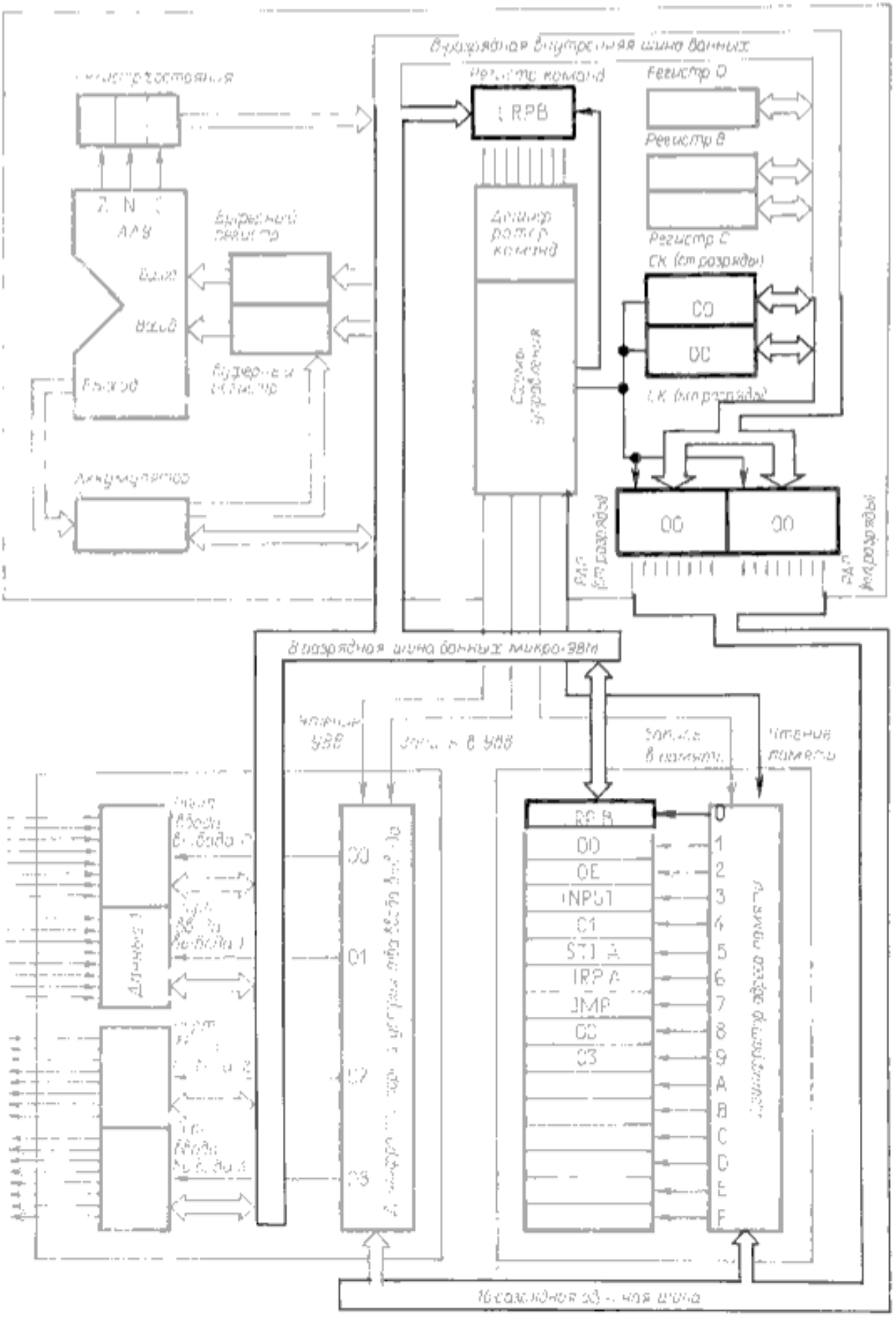


Рис. 7.4. Выполнение цикла выборки. В регистр команд микропроцессора загружается команда LRP B.

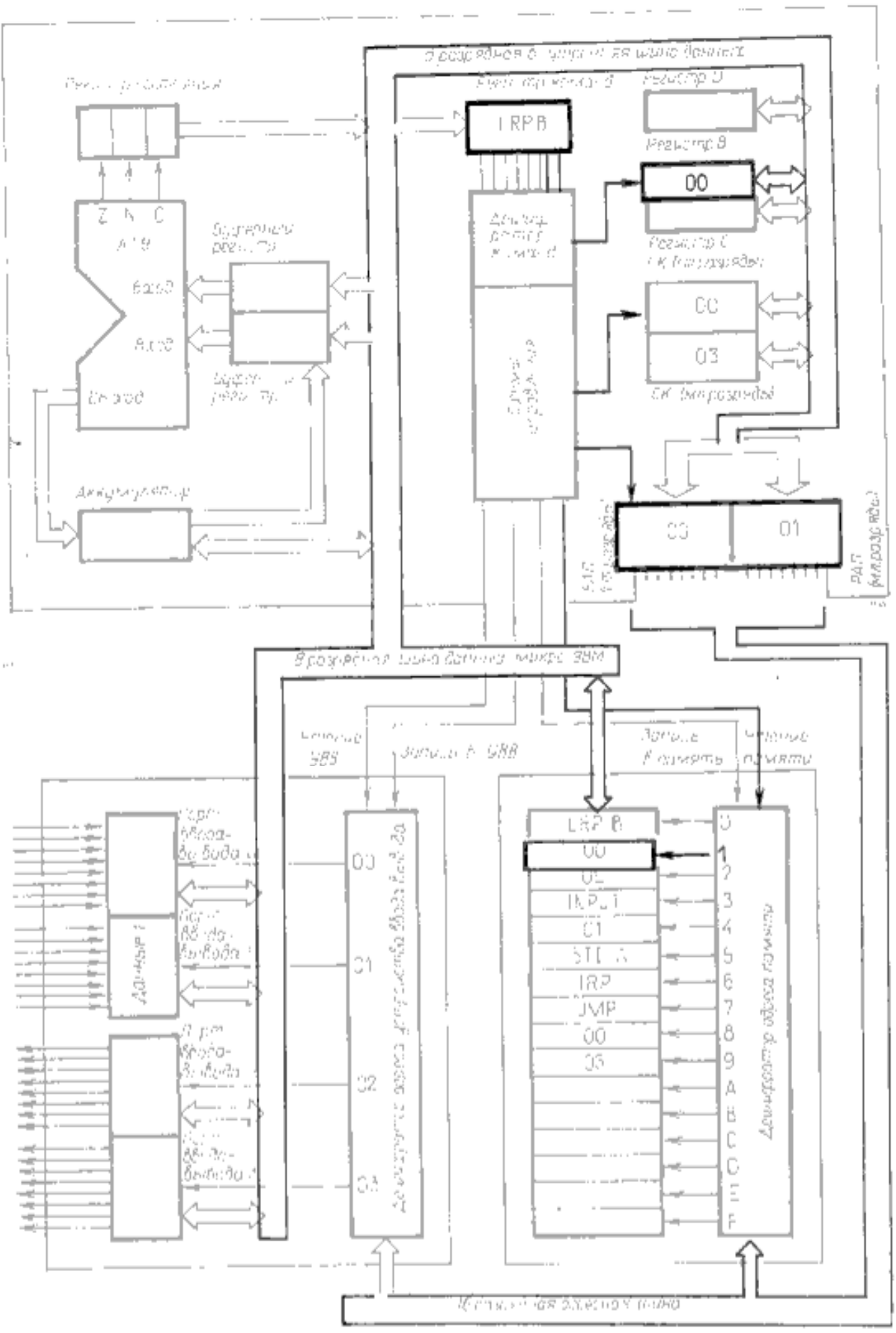


Рис. 7.5. Реализация первого цикла фазы выполнения. Данные из области памяти 0001 загружаются в регистр B.

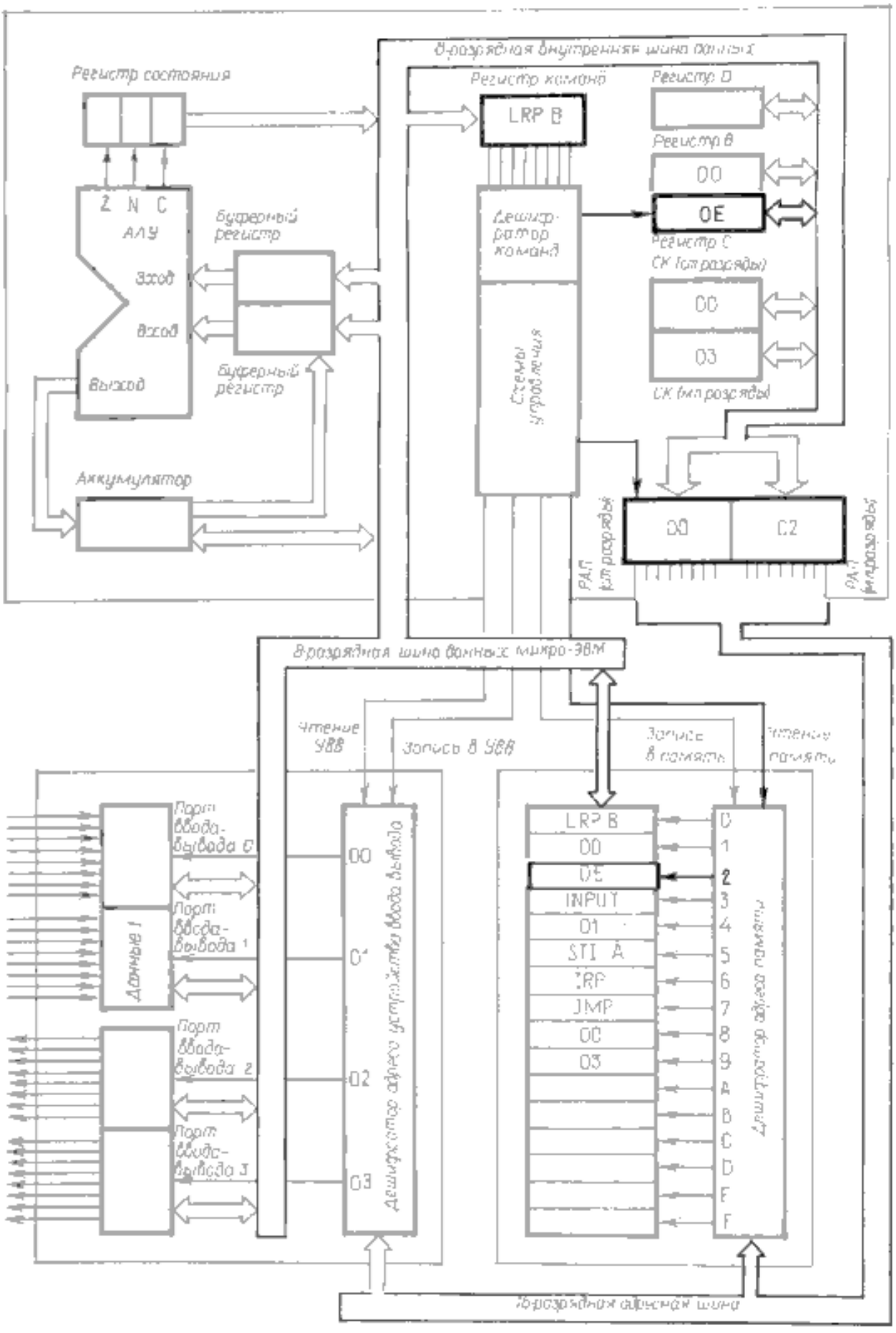


Рис. 7.6. Реализация второго цикла фазы выполнения. Данные из области памяти 0002 загружаются в регистр С, т.е. завершается выполнение команды ЗАГРУЗКА РЕГИСТРОВОЙ ПАРЫ

На этом заканчивается выполнение команды ЗАГРУЗКА РЕГИСТРОВОЙ ПАРЫ НЕПОСРЕДСТВЕННАЯ. В пару регистров ВС загружены данные 000Е. Устройству управления известно, что оно полностью завершило выполнение команды. Теперь оно начинает реализацию нового цикла выборки.

На рис. 7.7–7.9 показаны этапы выполнения микропроцессорной системой трехцикловой 2-байтовой команды ВВОД. По команде ВВОД данные пересылаются из порта ввода 01 в аккумулятор. Наш микропроцессор не располагает возможностью непосредственной пересылки данных из порта ввода в область памяти.

На рис. 7.7 представлен цикл выборки этой команды. Содержимое счетчика команд загружается в регистр адреса памяти. Из этого регистра адрес (указывающий на область 0003) подается на адресную шину памяти. Дешифратор адреса памяти декодирует адрес. Устройство управления вырабатывает сигнал «Чтение памяти», являющийся для памяти указанием вывести содержимое области памяти 0003 на шину данных. Затем в регистр команд микропроцессора пересылается команда ВВОД. Тем самым завершается цикл выборки команды.

Рис. 7.8 иллюстрирует первую половину фазы выполнения команды. Команда ВВОД декодируется. Так как она имеет длину 2 байт, устройство управления сразу же осуществляет положительное приращение счетчика команд, в результате чего он указывает на область памяти 0005, являющуюся местом хранения в памяти следующей команды.

В связи с тем что выполняется именно 2-байтовая команда, устройство управления побуждает регистр адреса памяти указывать на соседнюю область памяти 0004. Регистр адреса памяти помещает этот адрес на адресную шину. Дешифратор адреса памяти декодирует этот адрес. Импульсный сигнал «Чтение памяти» сообщает о том, что необходимо вывести содержимое области 0004 на шину данных микро-ЭВМ. Данные (число 01) загружаются в младший байт регистра адреса памяти по сигналу ф2 генератора тактовых импульсов микропроцессора. Обратите внимание, что за время рассматриваемого цикла выполнения в регистре адреса памяти размещаются поочередно два разных числа: сначала адрес памяти, затем данные 01.

На рис. 7.9 показано выполнение третьего цикла второй команды программы, т.е. второй половины фазы выполнения этой команды. На адресных линиях находится число XX01 (символы X означают, что значение старшего байта несущественно). Адрес 01 передается по восьми линиям младшего байта шины. Устройство управления вырабатывает на этот раз импульсный сигнал «Чтение устройства ввода-вывода». Это означает, что в данном случае вместо логических схем управления памятью используются логические схемы управления вводом-выводом. Дешифратор устройства ввода-вывода получает импульсный сигнал «Чтение устройства ввода-вывода», в соответствии с которым схемы ввода-вывода помещают данные, находящиеся в данный момент в порте 01, на шину данных микро-ЭВМ. Данные из порта 01 пересылаются в аккумулятор микропроцессора.

На этом заканчивается выполнение второй команды. Содержимое порта 01 (первое слово данных) находится теперь в аккумуляторе.

Рис. 7.10 и 7.11 иллюстрируют выполнение микропроцессорной системой двухцикловой 1-байтовой команды ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ. По этой команде содержимое аккумулятора (первое слово данных) записывается в область памяти, на которую указывает регистровая пара ВС.

На рис. 7.10 представлен цикл выборки этой команды. Содержимое счетчика команд загружается в регистр адреса памяти. Из регистра адреса памяти адрес 0005 выводится на адресную шину. Дешифратор адреса получает его и декодирует. Устройство управления посылает импульсный сигнал «Чтение памяти». Содержимое области памяти 0005 (команда ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ) поступает на шину данных и пересылается в регистр команд микропроцессора.

На рис. 7.11 показано состояние микро-ЭВМ после выполнения данной команды. Расшифровав команду ЗАПИСЬ В ПАМЯТЬ, устройство управления выясняет, что это 1-байтовая команда, и увеличивает значение счетчика команд на 1. Счетчик команд указывает на следующую команду.

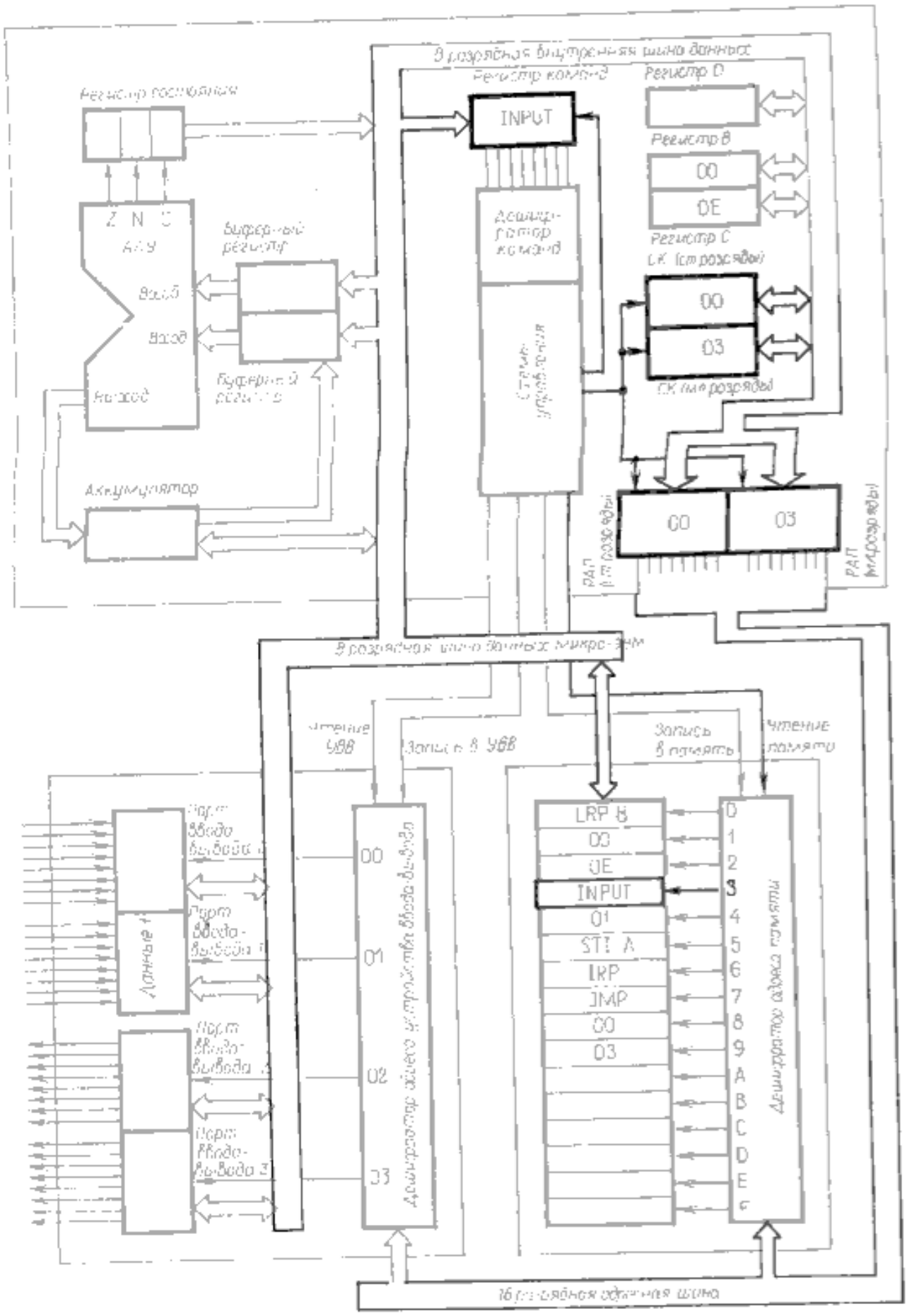


Рис. 7.7. Выборка команды ВВОД – начало выполнения второй команды.

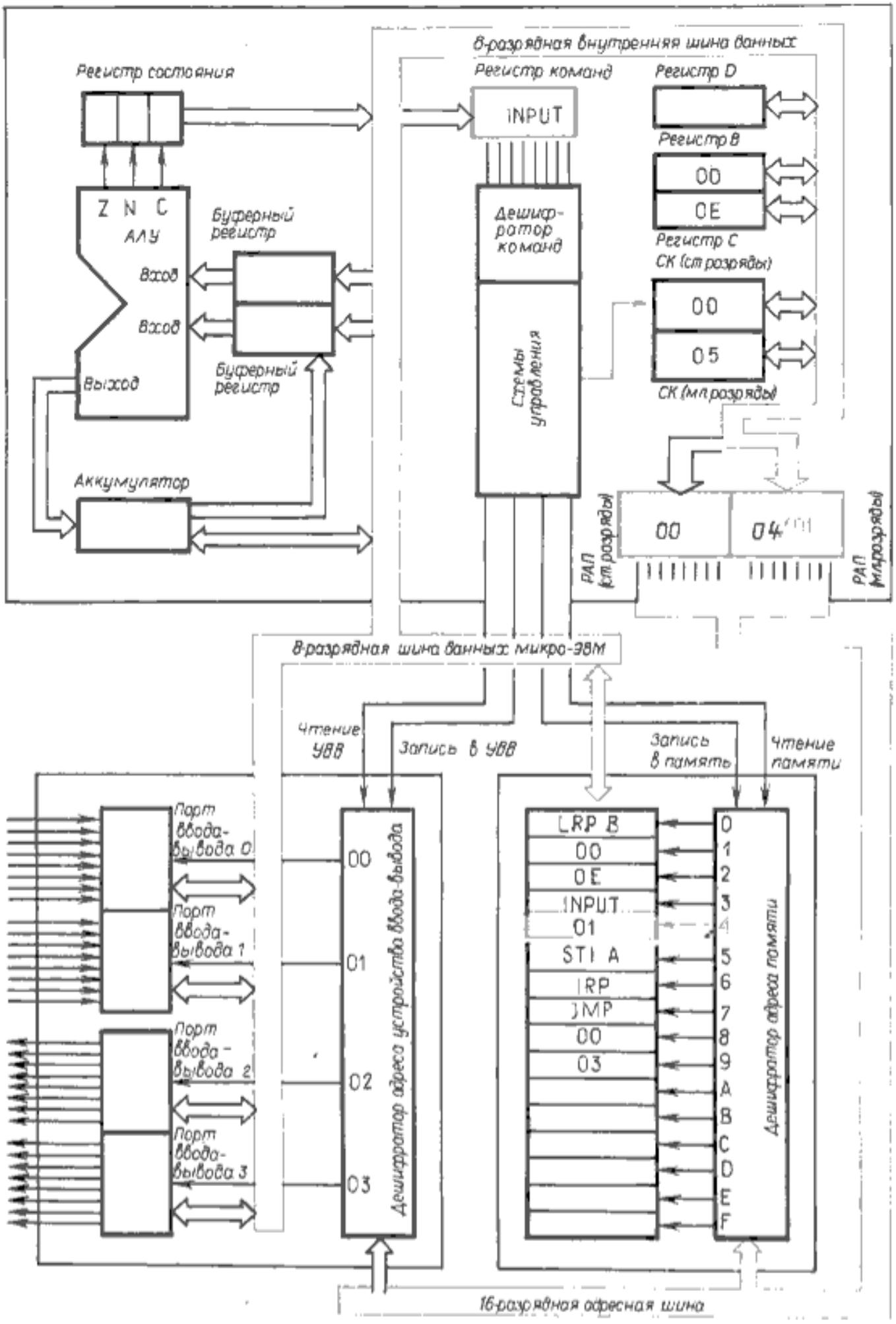


Рис. 7.8. Реализация первого цикла фазы выполнения команды ВВОД. (Адрес порта ввода-вывода загружается в младший байт регистра адреса памяти.)

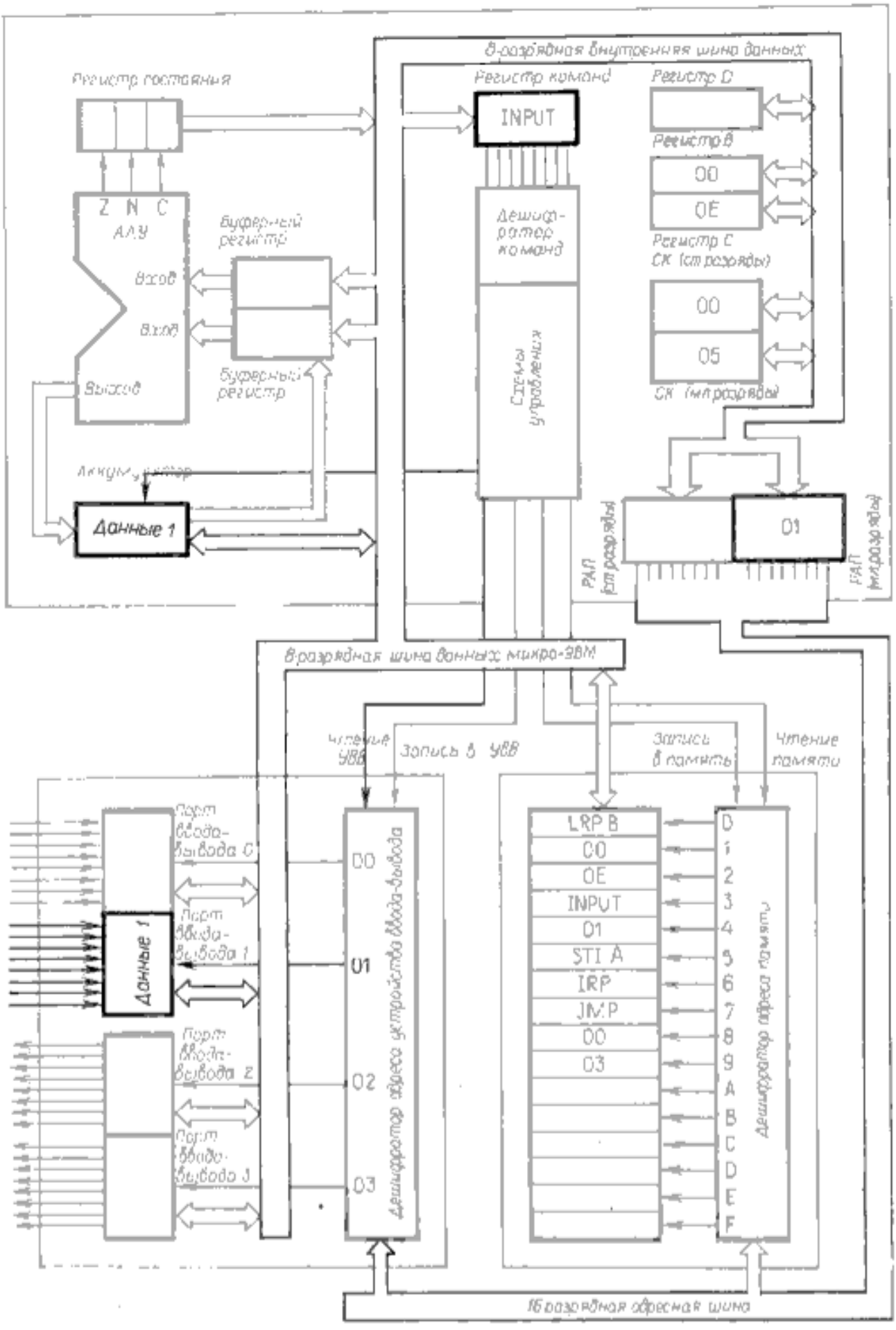


Рис. 7.9. Реализация второго цикла фазы выполнения команды. (Первая пересылка данных ввода-вывода; 8-разрядное слово данных – «Данные 1» – передается с 8 линий порта ввода-вывода 01 в аккумулятор.)

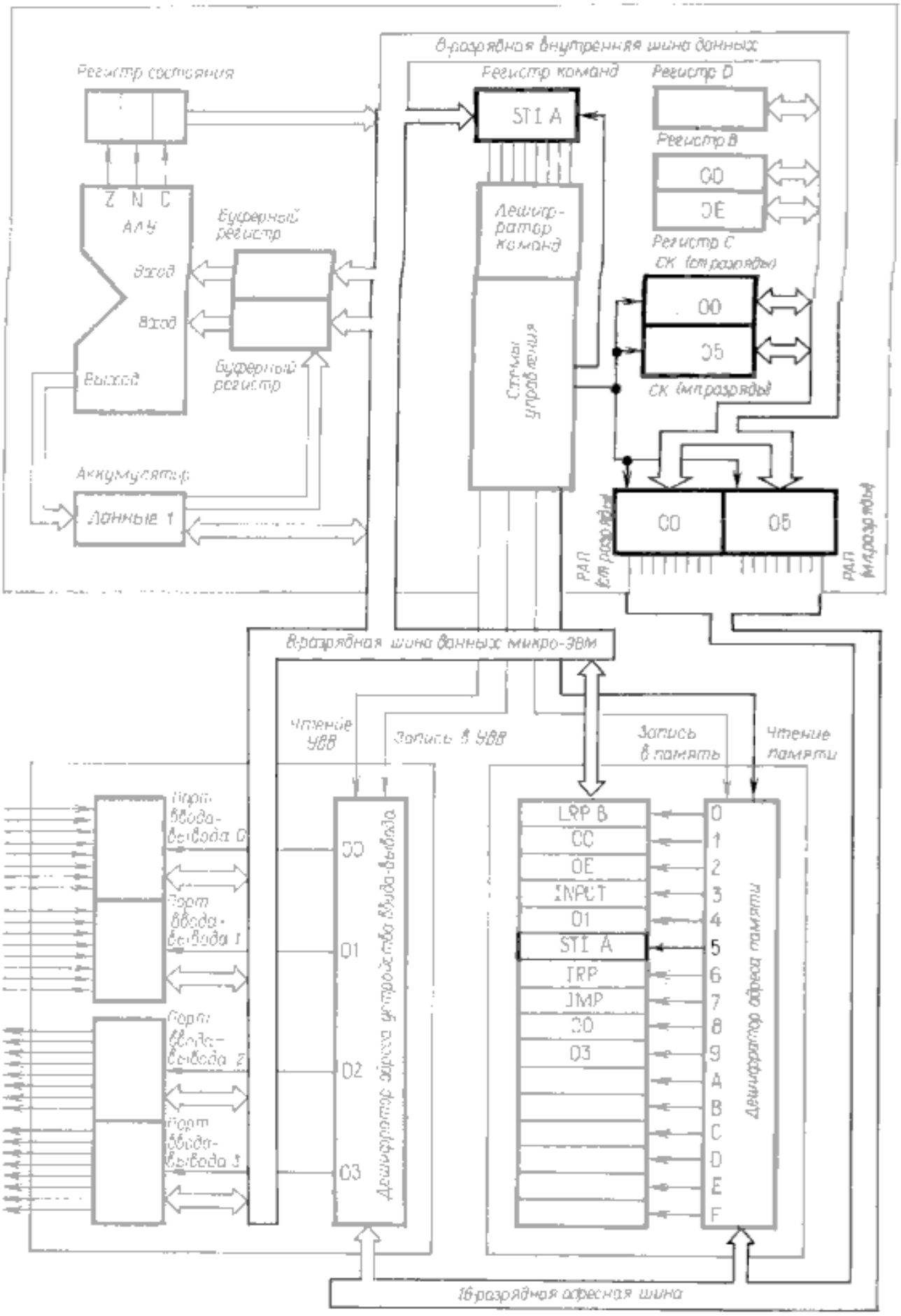


Рис. 7.10. Выборка команды ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ.

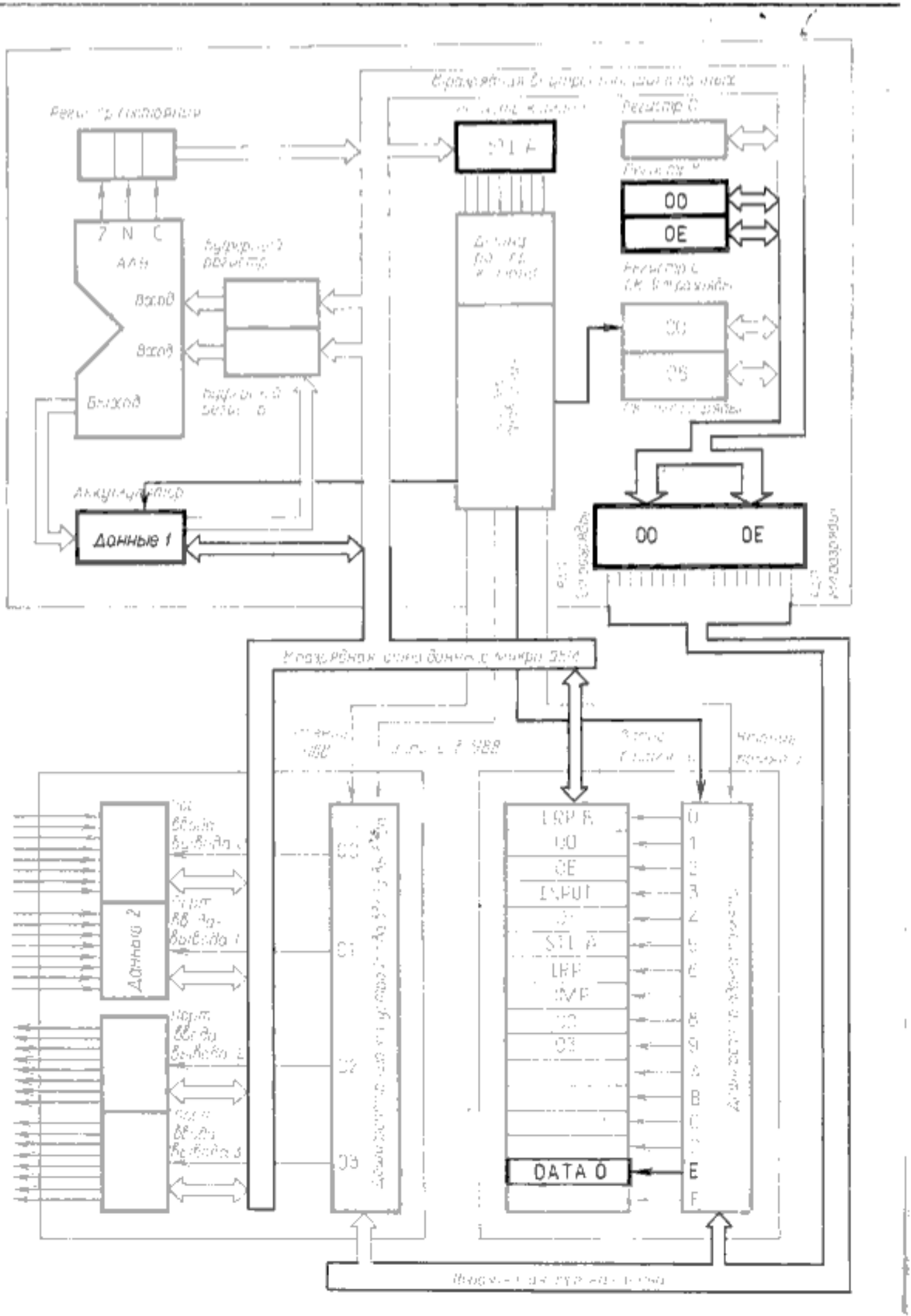


Рис. 7.11. Выполнение команды записи в память. (Содержимое аккумулятора – «Данные 1» – записывается в область памяти 000E. Адрес памяти поступает из регистровой пары BC.)

Содержимое регистровой пары BC (число 000E) пересылается в регистр адреса памяти. Из регистра адреса памяти эта адресная информация выводится на адресную шину. В то же время содержимое аккумулятора помещается на шину данных микропроцессора. Дешифратор адреса памяти декодирует адрес 000E. При поступлении сигнала «Запись в память» информация, имеющая место на шине данных, записывается в область памяти 000E. Теперь в области 000E содержится копия данных, находящихся в аккумуляторе. Это первое слово данных, считанное из порта ввода 01. Выполнение команды ЗАПИСЬ В ПАМЯТЬ завершается. Первое слово данных занесено в файл данных.

На рис. 7.12 и 7.13 представлены циклы выборки и выполнения команды ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ РЕГИСТРОВОЙ ПАРЫ BC. В результате выполнения этой команды регистровая пара указывает на следующую в порядке возрастания адресов область памяти. После того как эта команда выполнится в первый раз, содержимое регистровой пары изменится с 000E на 000F. Благодаря использованию такого приема организуется последовательный файл данных.

На рис. 7.12 показан цикл выборки данной команды. Содержимое счетчика команд загружается в регистр адреса памяти. Адрес 0006 выдается из этого регистра на адресную шину. В дешифратор адреса памяти поступают адрес и сигнал «Чтение памяти». Содержимое области памяти 0006 помещается на шину данных микро-ЭВМ. Команда ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ РЕГИСТРОВОЙ ПАРЫ пересылается в регистр команд.

Рис. 7.13 иллюстрирует цикл выполнения команды. Так как команда 1-байтовая, содержимое счетчика команд увеличивается только на 1. Счетчик команд указывает после этого на следующую команду.

В соответствии с текущей командой устройство управления заставляет микропроцессор осуществить положительное приращение пары регистров BC. Как следует из рисунка, содержимое этой пары регистров увеличилось на 1.

В данном случае с помощью одной команды производится положительное приращение содержимого 16-разрядного регистра. Перенос из старшего бита младшего байта в младший бит старшего байта, если он необходим, происходит автоматически. Это возможно, так как при выполнении рассматриваемой команды пара регистров функционирует как один 16-разрядный регистр.

Содержимое регистра адреса памяти в течение фазы выполнения команды не имеет значения, так как обращений к памяти в это время не происходит.

На рис. 7.14–7.16 показаны циклы выборки и выполнения команды перехода, имеющей длину 3 байт и выполняемой за три цикла. Это последняя команда в нашей программе. В результате ее выполнения управление опять передается команде ВВОД. Это означает, что после выполнения команды перехода программа готова вернуться к началу цикла и продолжить ввод очередных данных.

Рис. 7.14 иллюстрирует цикл выборки команды перехода. В регистр адреса памяти загружается содержимое счетчика команд. На адресную шину памяти подается адрес 0007. Дешифратор адреса декодирует этот адрес. При поступлении сигнала «Чтение памяти» содержимое области 0007 выводится из памяти на шину данных микро-ЭВМ. Это содержимое (команда перехода) загружается в регистр команд.

Первый цикл фазы выполнения команды и перехода показан на рис. 7.15. Во время реализации этого цикла в старший байт счетчика команд загружаются данные, содержащиеся во втором байте команды перехода. Как мы помним, команда перехода изменяет содержимое счетчика команд. Последний не будет указывать после ее выполнения на следующую по порядку команду программы. По завершении выполнения команды перехода он указывает на область памяти, адрес которой содержался в команде перехода. Именно в этой области находится команда, которую надо выполнить следующей, но не команда, которая хранится в области, следующей в порядке возрастания адресов.

Так как надо обратиться ко второму байту 3-байтовой команды, регистр адреса памяти задает сейчас область памяти 0008. Адрес 0008 поступает из регистра на адрес-

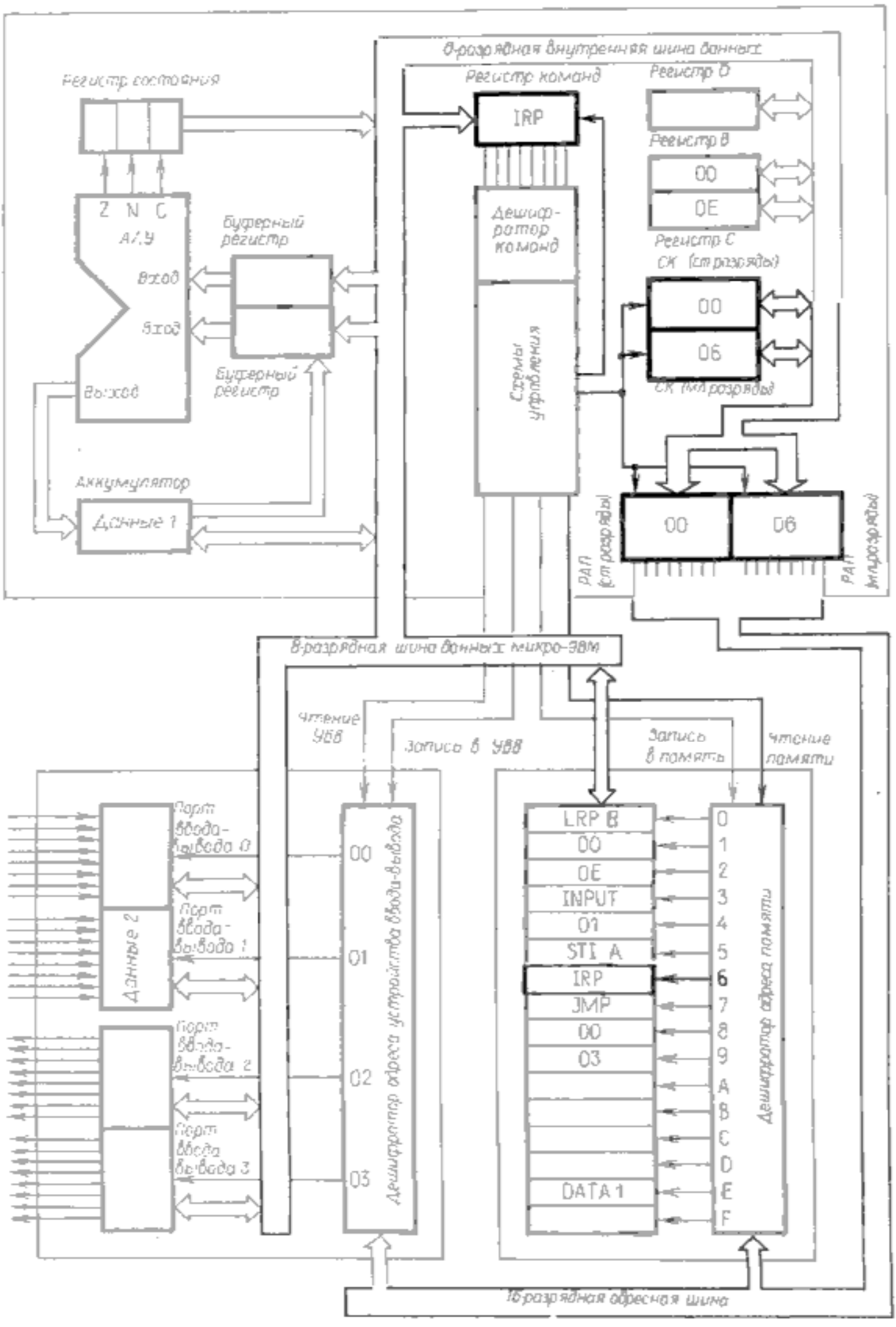


Рис. 7.12. Выборка команды ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ РЕГИСТРОВОЙ ПАРЫ. (Предыдущие данные «Данные 1» еще находятся в аккумуляторе и будут оставаться там до тех пор, пока в аккумулятор не поступят новые данные или его содержимое не будет сброшено.)

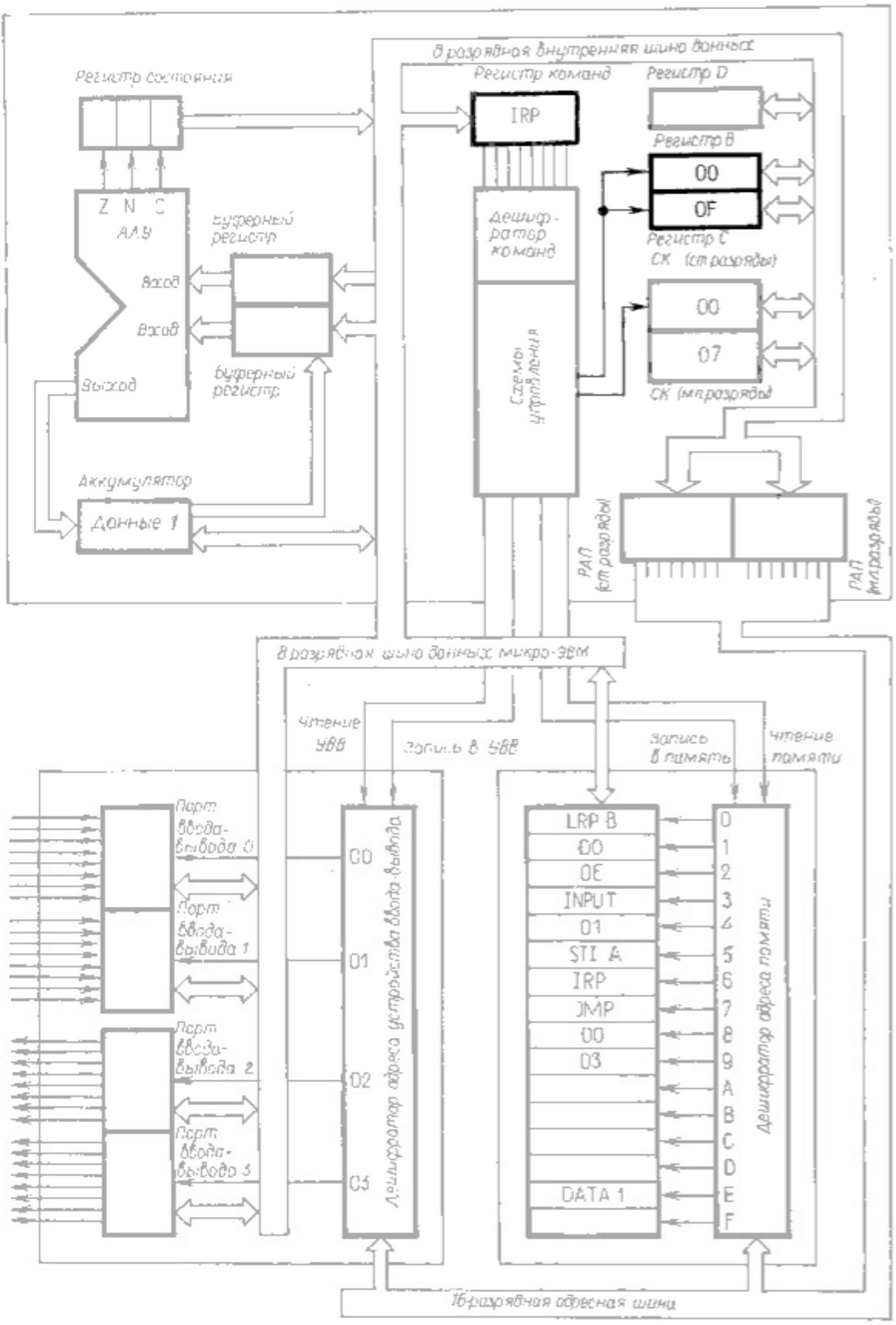


Рис. 7.13. Выполнение команды ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ РЕГИСТРОВОЙ ПАРЫ. (Содержимое регистровой пары BC – число 000E – увеличивается на 1 и становится равным 000F. Содержимое счетчика команд также увеличивается на 1, в результате чего он указывает на область памяти, содержащую команду JMP – ПЕРЕХОД.)

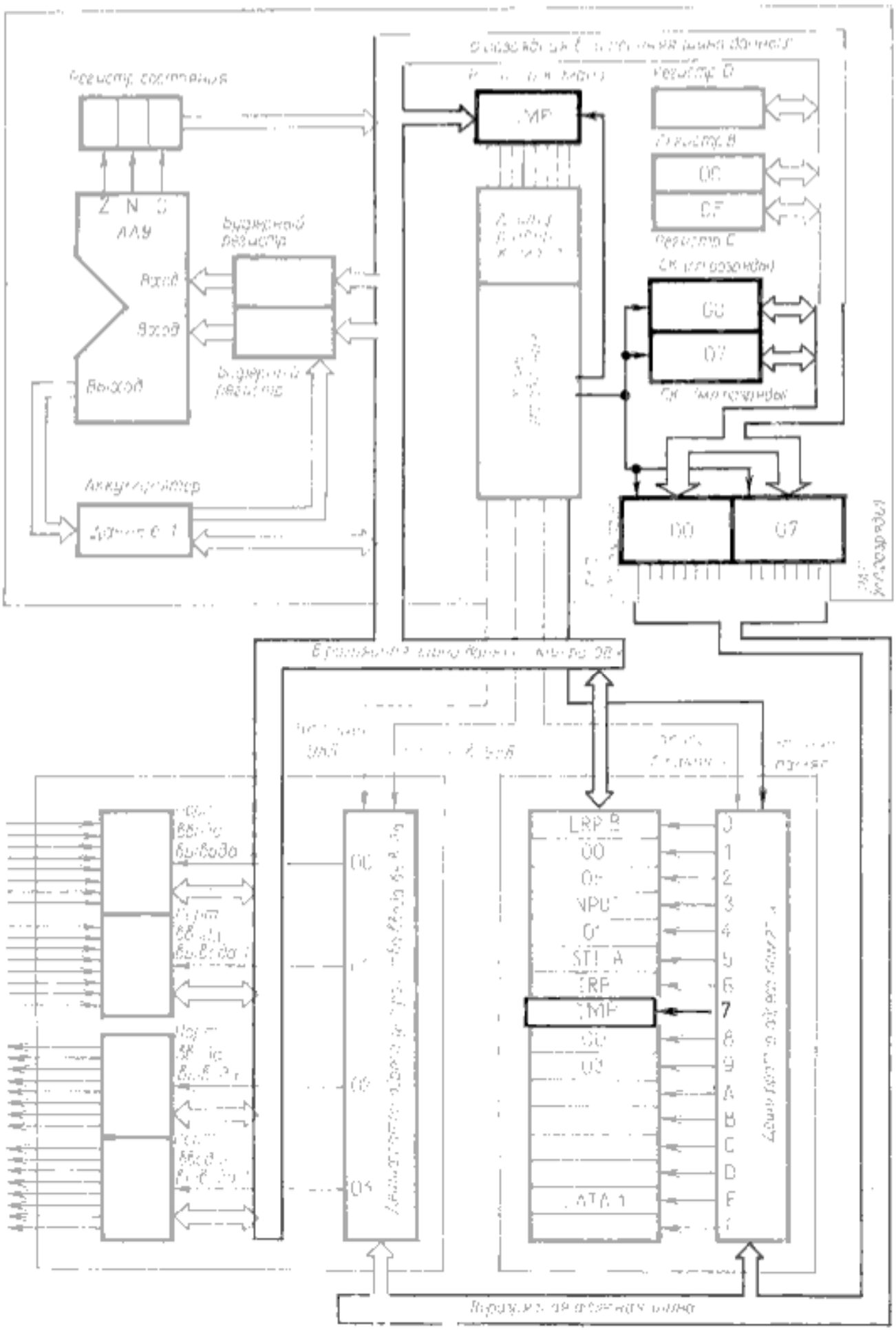


Рис. 7.4. Выборка команды перехода.

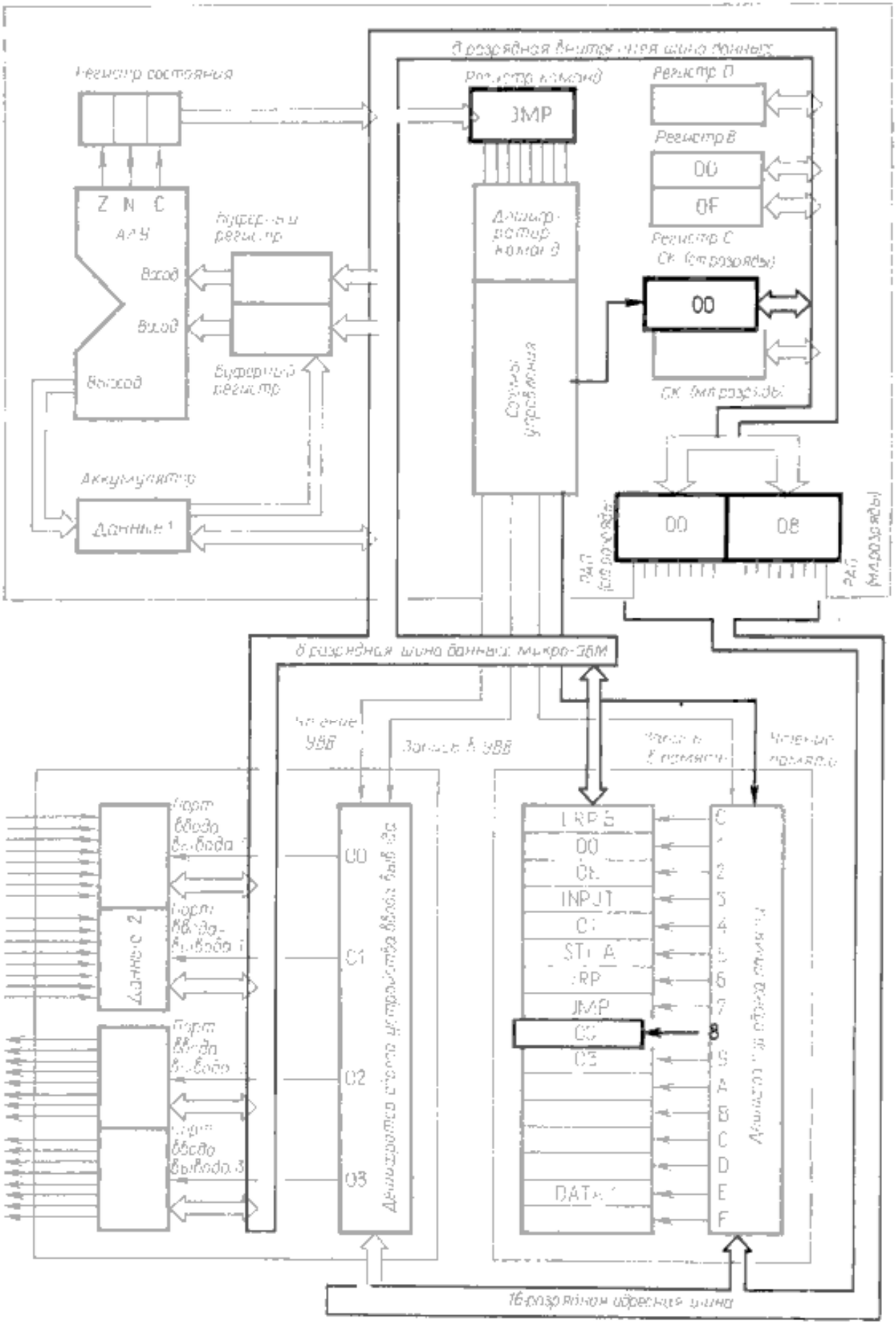


Рис 7.15. Реализация первого цикла фазы выполнения команды перехода. (Загрузка старшего байта счетчика команд.)

ную шину. Дешифратор адреса памяти декодирует этот адрес. По сигналу «Чтение памяти» содержимое области памяти 0008 (число 00) выводится на шину данных. Данные (число 00) пересылаются в старший байт счетчика команд. Старший байт счетчика команд содержит теперь число 00.

Второй цикл фазы выполнения иллюстрирует рис. 7.16. Так как мы имеем в данном случае дело с 3-байтовой командой, регистр адреса памяти указывает на ячейку 0009, где находится третий байт команды перехода. Адрес декодируется дешифратором адреса памяти. При поступлении сигнала «Чтение памяти» содержимое ячейки 0009 (число 03) выводится на шину данных и пересылается в младший байт счетчика команд. Теперь в младшем байте счетчика команд содержится число 03. Выполнение программы на этом заканчивается. Являющееся ее результатом содержимое счетчика команд будет использовано в начале выполнения следующей команды. Счетчик команд указывает на область памяти 0003.

Завершился полный цикл выполнения программы. Если не принимать специальных мер, такие циклы будут повторяться бесконечно. Рассмотрим еще несколько шагов программы, выполняемых до того, как произойдет останов.

На рис. 7.17 показан цикл выборки трехцикловой 2-байтовой команды ВВОД. За время выполнения этого цикла в регистр адреса памяти загружается содержимое счетчика команд. Из регистра снова выводится на адресную шину адрес области 0003. Дешифратор декодирует этот адрес. После получения сигнала «Чтение памяти» на шину данных микро-ЭВМ поступает содержимое области 0003 (команда ВВОД). Так как имеет место цикл выборки, это содержимое загружается в регистр команд.

Второй цикл (рис. 7.18) представляет собой первую половину фазы выполнения. Регистр адреса памяти указывает на второй байт команды. Чтобы счетчик команд указывал на адрес следующей команды, содержимое его подвергается двукратному положительному приращению, так как выполняемая сейчас команда ВВОД имеет длину 2 байт. Регистр адреса памяти указывает сейчас на область памяти 0004. По сигналу «Чтение памяти» содержимое ячейки 0004 выдается из памяти на шину данных. Эти данные заносятся по тактовому сигналу ф2 в младший байт регистра адреса памяти.

Третий цикл команды ВВОД, представляющий собой вторую половину фазы выполнения, показан на рис. 7.19. Из регистра адреса памяти адрес 01 выводится на адресную шину. На этот раз, однако, сигналы «Чтение памяти» и «Запись в память» не поступают. Вместо этого подается сигнал «Чтение устройства ввода-вывода» По его получении данные, содержащиеся в порте входных данных 01, помещаются на шину данных микро-ЭВМ. Затем они загружаются в аккумулятор.

Выполнение команды на этом заканчивается. Новые данные (второе слово данных) находятся теперь в аккумуляторе. Завершилось вторичное считывание данных из порта ввода-вывода 01. Как показано на рис. 7.20, данные аккумулятора пересылаются во вторую область файла данных в памяти. Пересылка эта осуществляется с помощью 1-байтовой двухцикловой команды ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ.

Во время цикла выборки этой команды содержимое счетчика команд загружается в регистр адреса памяти. Переданный по адресной шине адрес 0005 декодируется дешифратором. После поступления импульса «Чтение памяти» содержимое области памяти 0005 (команда ЗАПИСЬ В ПАМЯТЬ) выводится на шину данных и пересылается в регистр команд.

В ходе цикла выполнения команды ЗАПИСЬ В ПАМЯТЬ (рис. 7.21) содержимое счетчика команд увеличивается на 1 и указывает теперь на следующую команду.

Содержимое регистровой пары ВС (000F) пересылается в регистр адреса памяти. Это число подается из регистра адреса памяти на адресную шину памяти. Содержимое аккумулятора (второе слово данных) помещается на шину данных микро-ЭВМ. При поступлении сигнала «Запись в память» эти данные с шины записываются в область памяти 000F.

Таким образом, произведена запись в память второго слова данных из порта 01. Оно находится во второй области памяти формируемого файла данных

Следующей командой является команда

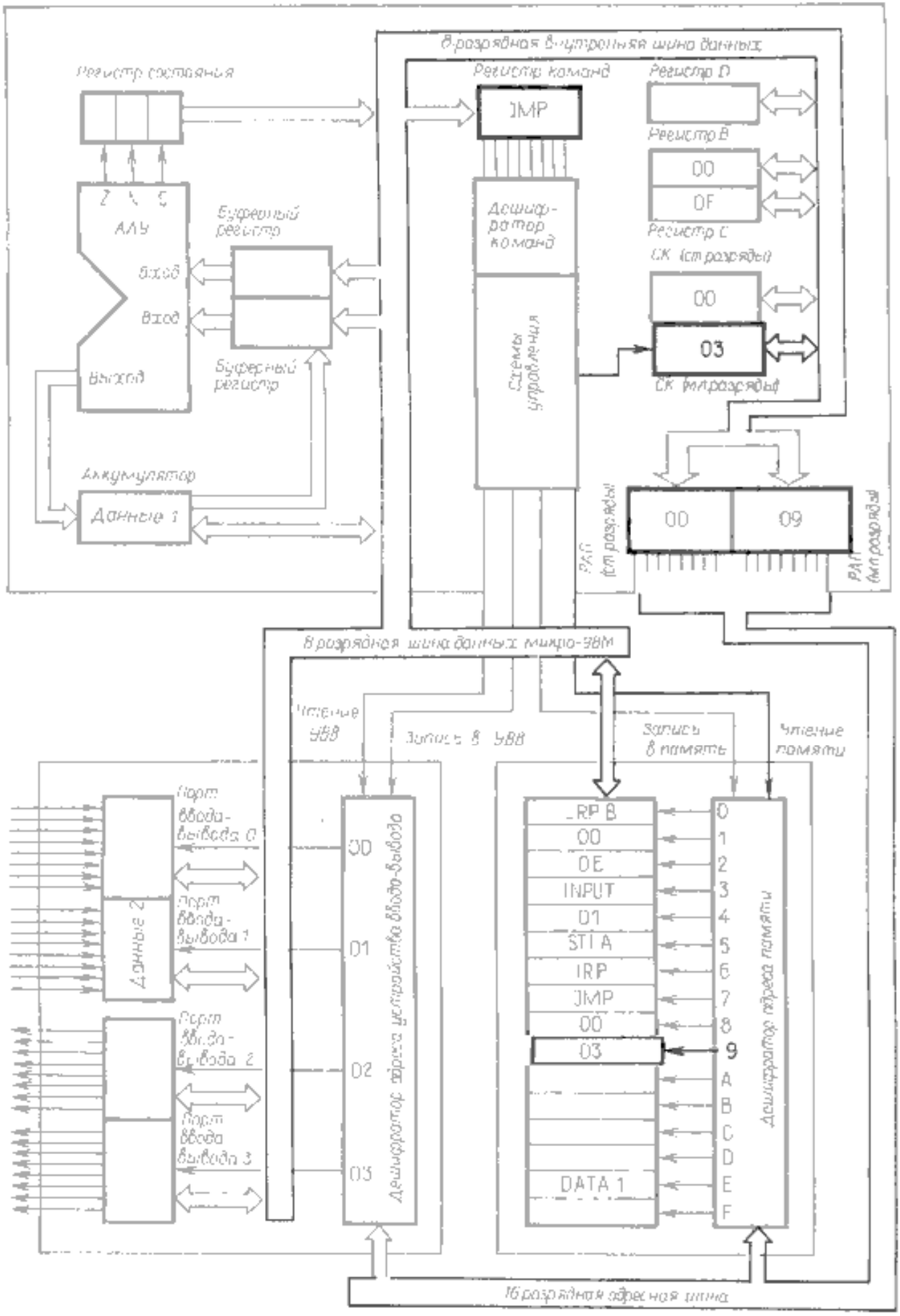


Рис. 7.16. Реализация второго цикла фазы выполнения команды перехода. (Загрузка младшего байта счетчика команд, который указывает теперь на область памяти 0003_{16}. Микропроцессор извлечет следующую команду из этой области, а не из области, следующей за той, в которой хранится команда JMP.)

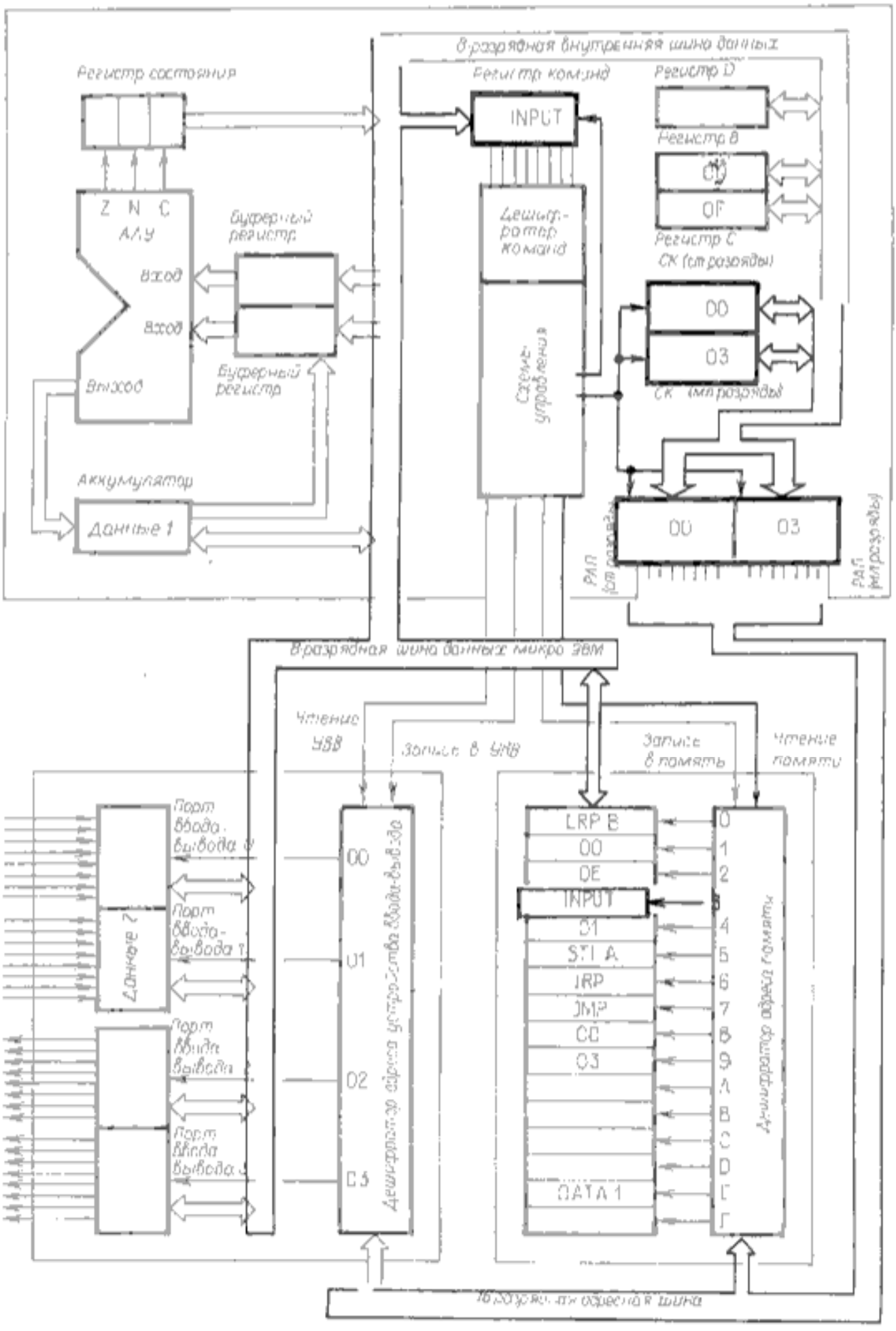


Рис. 7.17. Вторая выборка команды ВВОД. В программе произошел переход, в результате которого осуществляется дальнейший прием данных.

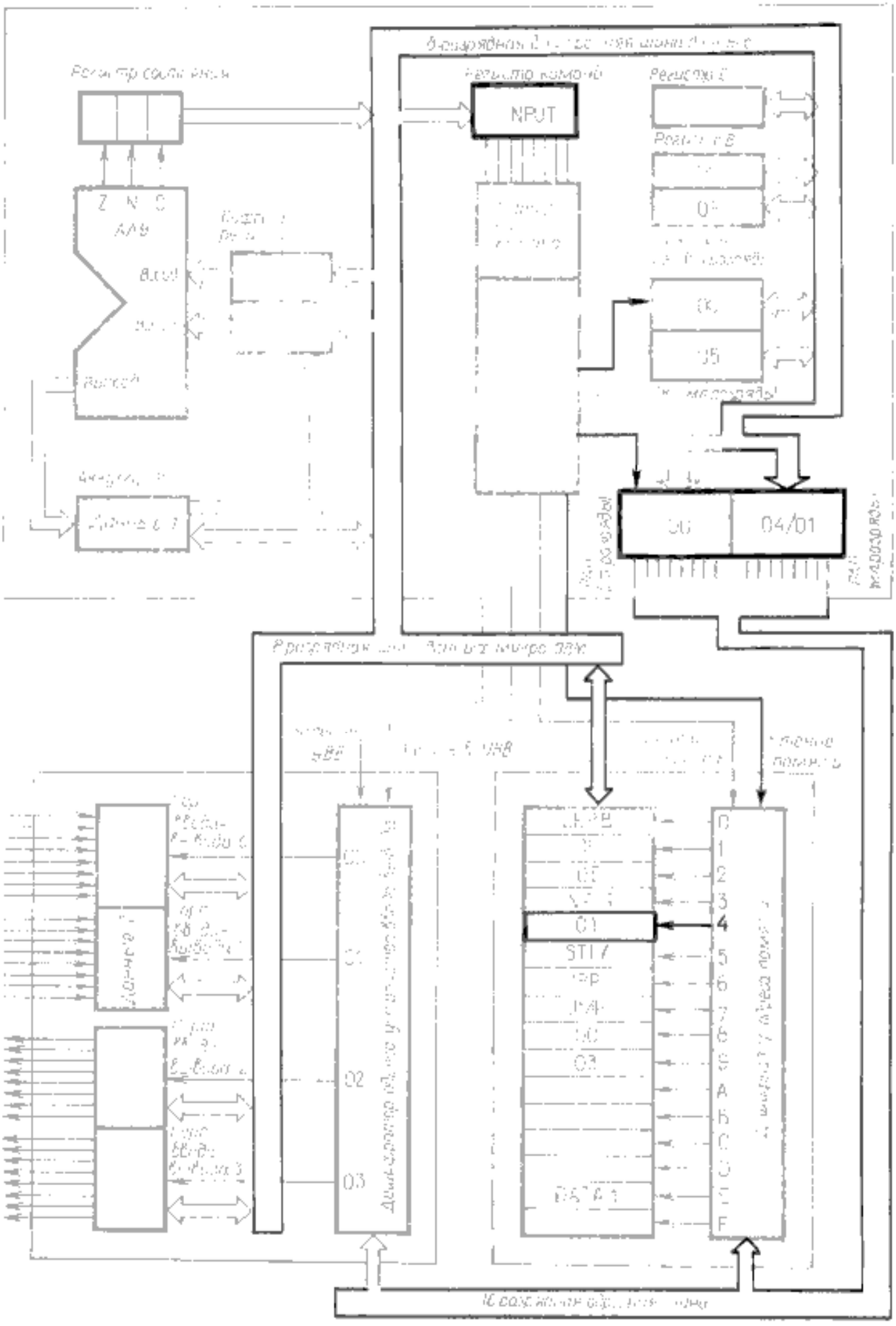


Рис. 7.18. Реализация первого цикла фазы выполнения. (Загрузка адреса порта ввода-вывода в младший байт регистра адреса памяти.)

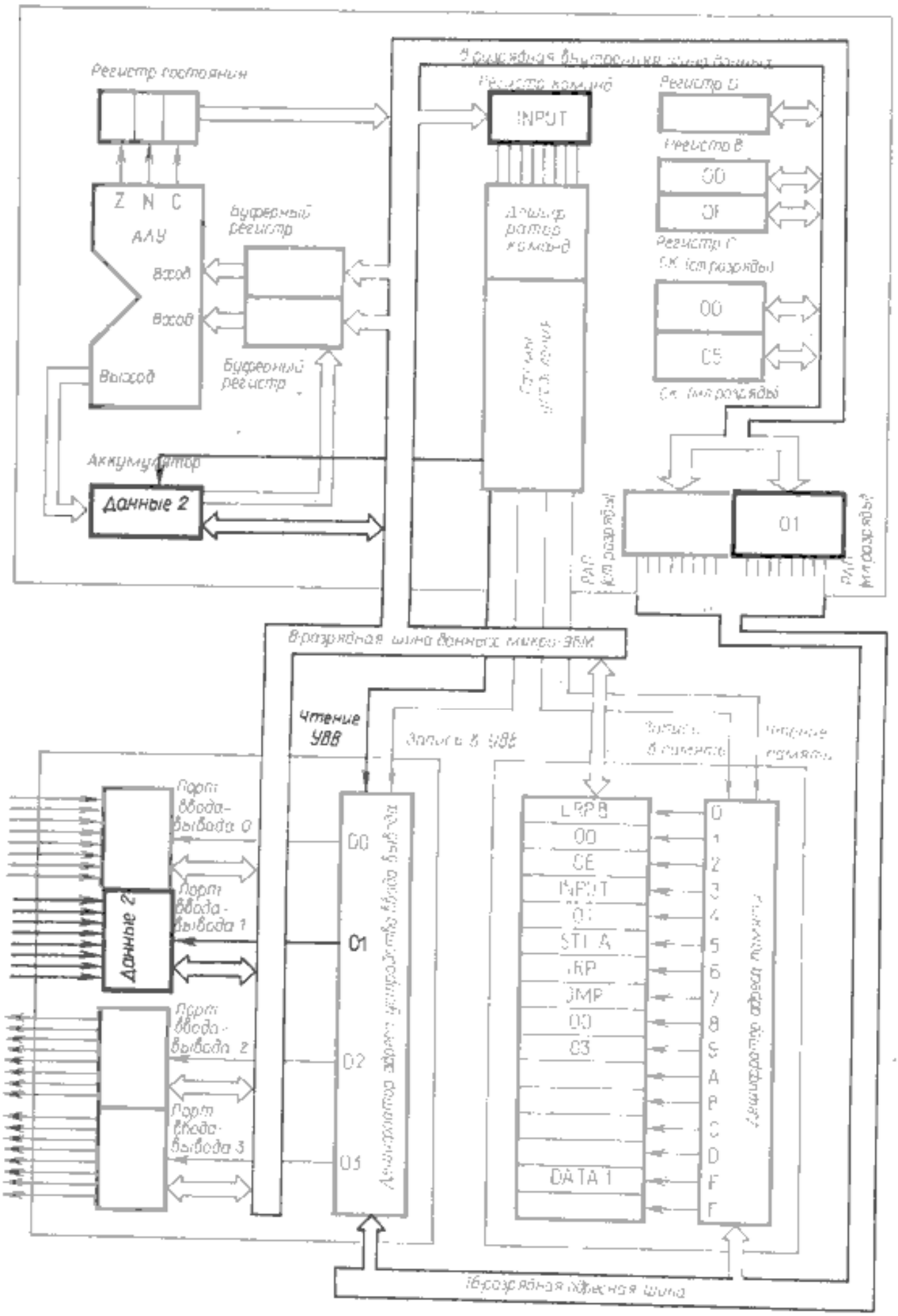


Рис. 7.19. Реализация второго цикла фазы выполнения. (Слово данных – «Данные 2» – пересылается из порта ввода-вывода 01 в аккумулятор. При его загрузке в аккумуляторе уничтожаются ранее находившиеся там «Данные 1».)

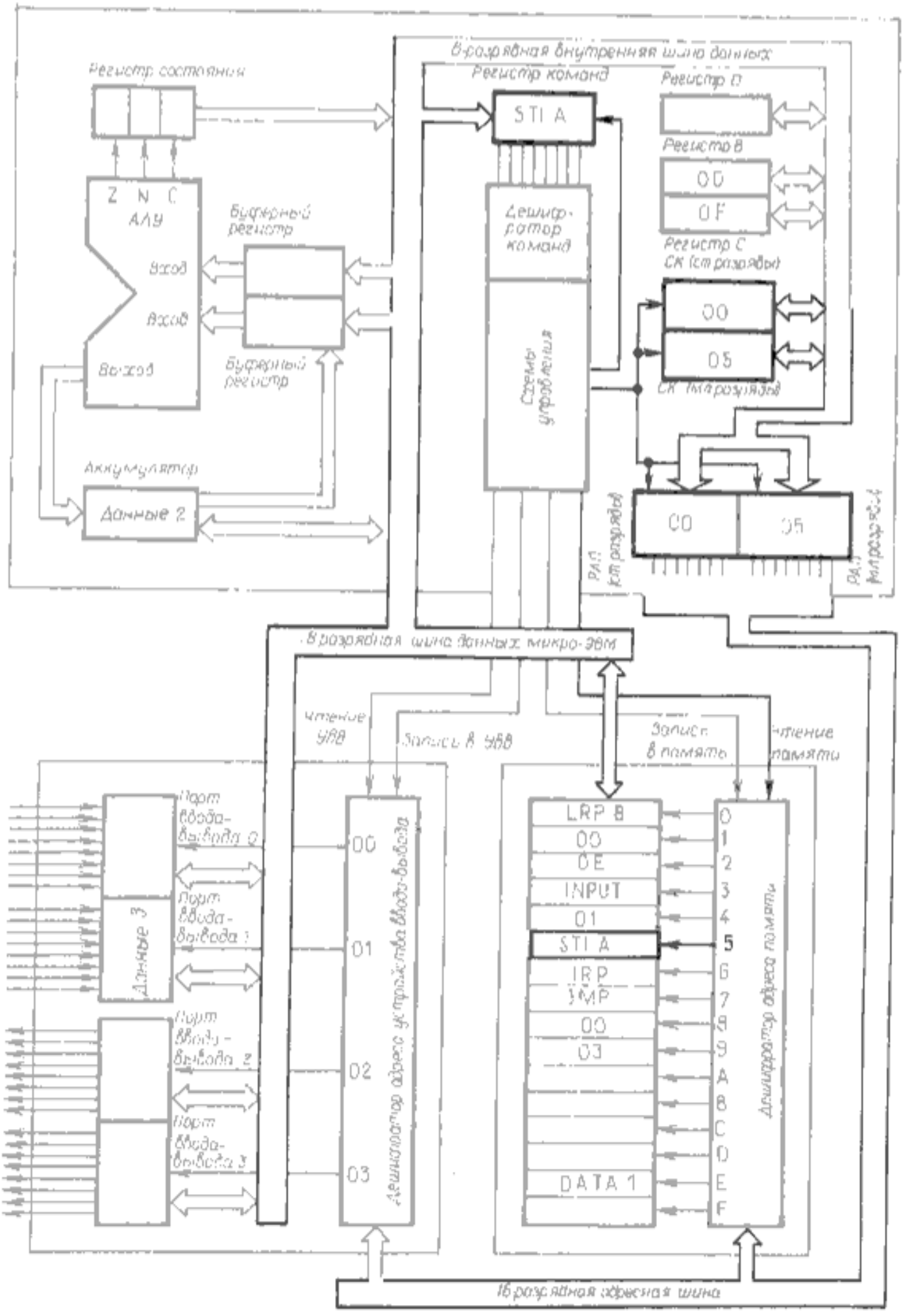


Рис. 7.20. Вторая выборка команды ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ.

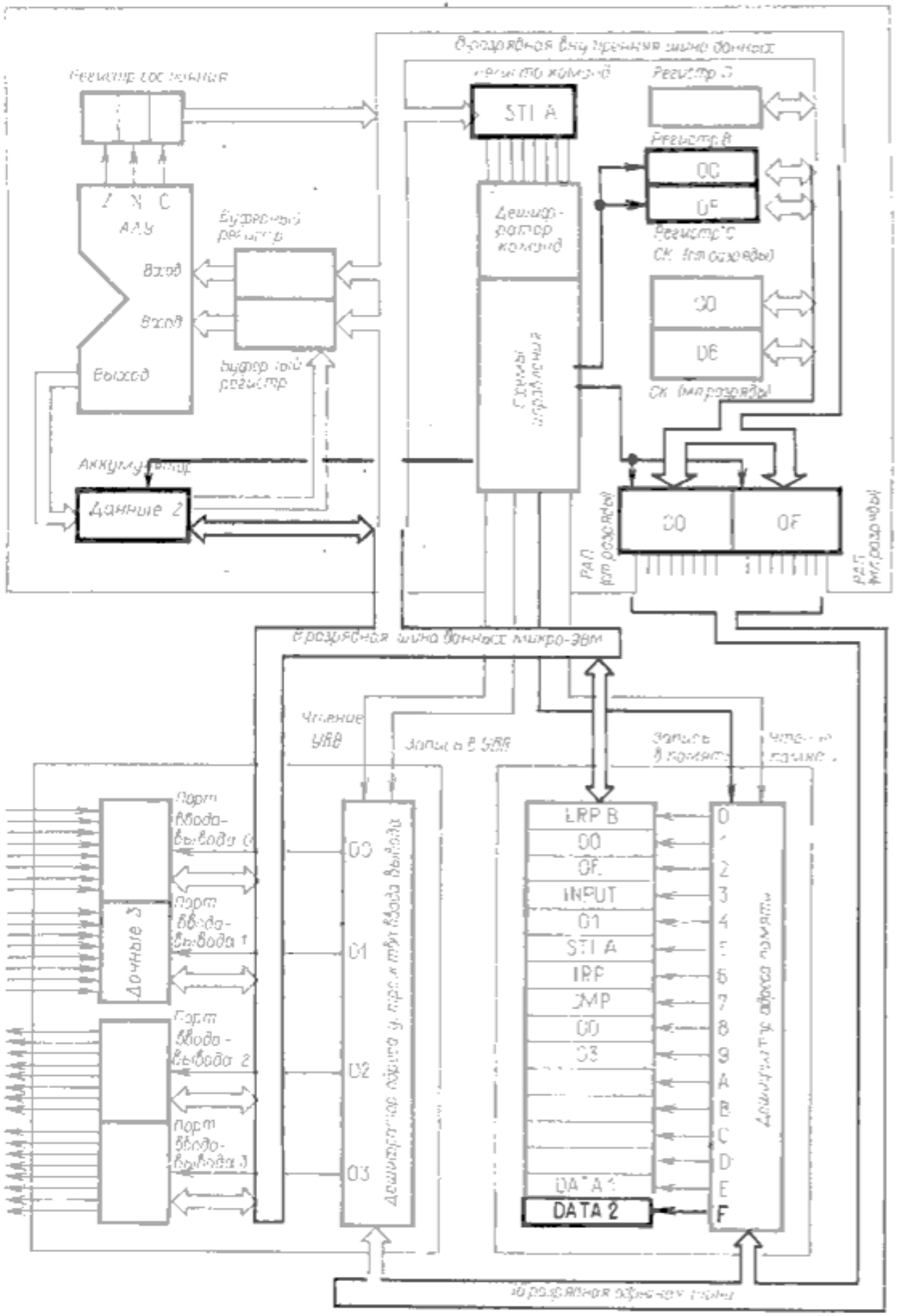


Рис. 7.21. Цикл выполнения второй команды ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ. («Данные 2» заносятся по этой команде в область памяти 000F. В регистр адреса памяти было загружено содержимое регистровой пары BC. Это значение, над которым выполнено положительное приращение, указывает на новую область.)

ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ РЕГИСТРОВОЙ ПАРЫ BC. Это действие выполняется снова, чтобы регистровая пара указывала на третью область файла данных, т. е. область памяти 0010.

Описанный процесс повторяется до тех пор, пока не произойдет останов микро-ЭВМ. Формируемый таким образом последовательный файл данных может быть сколь угодно длинным, единственное ограничение состоит в наличии свободной памяти. Как было отмечено выше, в рассмотренной программе не предусмотрено внутренних средств останова.

Задания для самопроверки

7. В рассмотренном выше примере программа выполнялась в следующей последовательности:

0000 0003 0005 0006 0007 0003 0005 0006 0007.

Почему программа не выполнялась в последовательности

0000 0001 0002 0003 и т. д.?

8. Куда и с какой целью данные из адресуемой области памяти пересылаются по шине данных в ходе цикла выборки команды?

9. При выполнении команды ЗАПИСЬ В ПАМЯТЬ используется больше циклов чтения информации из памяти, чем циклов записи информации в память. Почему?

10. Изобразите блок-схему программы, которая осуществляет запись файла данных в области памяти с адресами 0100–01FF. Программа должна помещать число 00 в область 0100, число 01 в область 0101, число 02 в область 0102 и т. д. Укажите, какие команды пересылки данных будут использованы при написании такой программы.

11. Почему при выполнении команды ВВОД, имеющейся в составе рассмотренного примера программы, микро-ЭВМ не считывает число 00 из области памяти 0001?

12. Если не принять специальных мер для останова рассмотренной в примере программы, то каким образом прекратится ее выполнение?

Упражнения

7.1. Какие функции выполняют команды пересылки данных? Содержимое каких областей участвует в их реализации?

7.2. Почему команды пересылки данных могут быть также названы командами «копирования данных»?

7.3. Определите команды, которые следовало бы использовать для выполнения перечисленных ниже действий:

а) переслать данные в аккумулятор из конкретной области памяти, на которую указывают второй и третий байты команды;

б) записать копию содержимого аккумулятора в область памяти, на которую указывает содержимое регистровой пары BC;

в) загрузить регистр B данными, содержащимися во втором байте команды;

г) переслать данные из регистра C в регистр B;

д) записать в область памяти FFAE копию содержимого аккумулятора;

е) загрузить содержимое аккумулятора в регистр D.

7.4. Опишите действия, выполняемые приведенными ниже командами:

а) LDA B,04
б) MOV A,C
в) STA E6B4
г) LRP B,501C
д) LDI A
е) LDA C,00
ж) MOV D,B
з) STI A
и) LDD D,00A1
к) STA 0000
л) LDA A,FF
м) LRP B,FFFF

7.5. Напишите короткую программу для пересылки данных из области памяти 001F в область 007E. Копия тех же данных должна остаться в регистре C.

Ответы на вопросы заданий для самопроверки

1. в.

2. По команде LDA A в аккумулятор загружаются данные, содержащиеся во втором байте команды. При выполнении команды LDD A в аккумулятор загружаются данные из области памяти, на которую указывают второй и третий байты команды. Выполнение команды LDD A длится на 1 микроцикл дольше, чем команды LDA A.

3. Команда LRP B,Данные выполняет те же действия, что и последовательность команд

LDA B,Данные и LDA C,Данные. Во втором случае затраты памяти относительно больше.
4. С помощью команды STA A в память записывается содержимое аккумулятора. По команде LDD A в аккумулятор загружаются данные из памяти. Это две обратные задачи.
5. Необходимо использовать команды MOV A,B и STA A,01FF. Данные пересылаются в аккумулятор, из которого помещаются в память с помощью команды ЗАПИСЬ В ПАМЯТЬ. См. рис. 7.22.

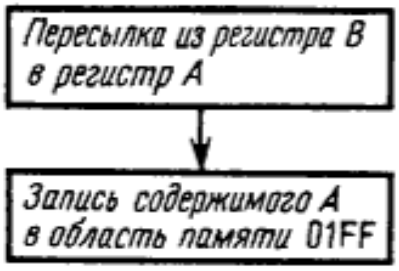


Рис. 7.22. Блок-схема алгоритма к ответу на вопрос п. 5 заданий для самопроверки.

6. См. рис. 7.23.

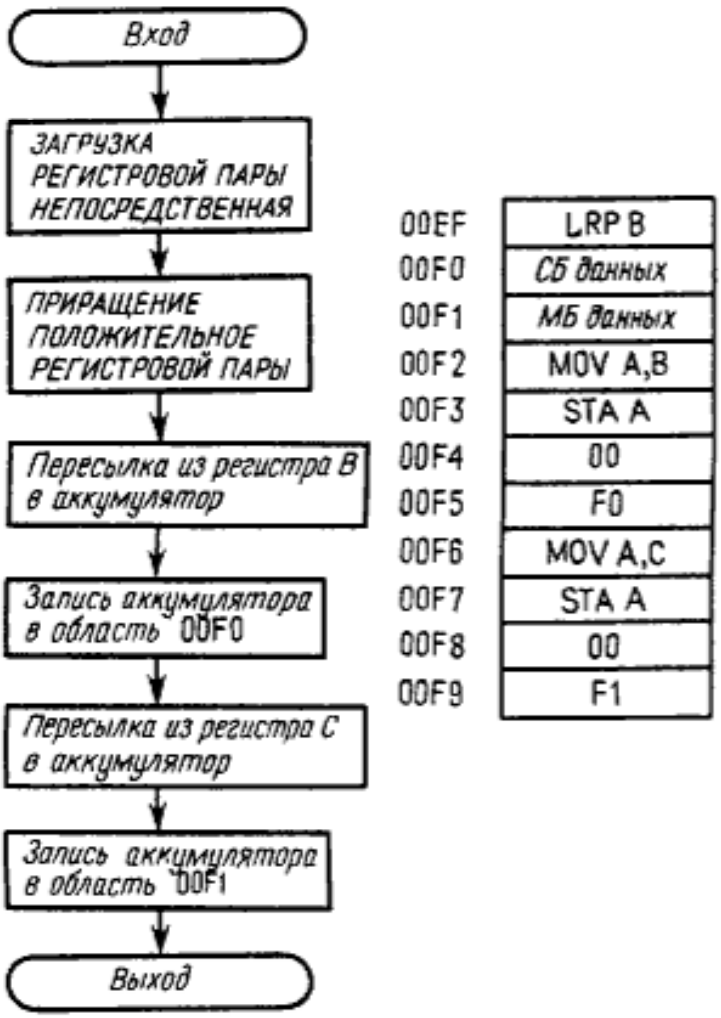


Рис. 7.23. Законченная блок-схема алгоритма и схема распределения памяти к ответу на вопрос п. 6 заданий для самопроверки.

7. Потому что в ходе выполнения программы происходят последовательные переходы к очередным командам, а не просто к последовательности байтов памяти.
8. В регистр команд. Это делается для того, чтобы дешифратор команд и устройство управления могли реализовать выполнение команды.
9. При выполнении команды ЗАПИСЬ В ПАМЯТЬ циклы чтения информации из памяти имеют место при выборке из памяти команды и 2 байт адреса. Для записи данных в память используется один цикл записи.
10. См. рис. 7.24.

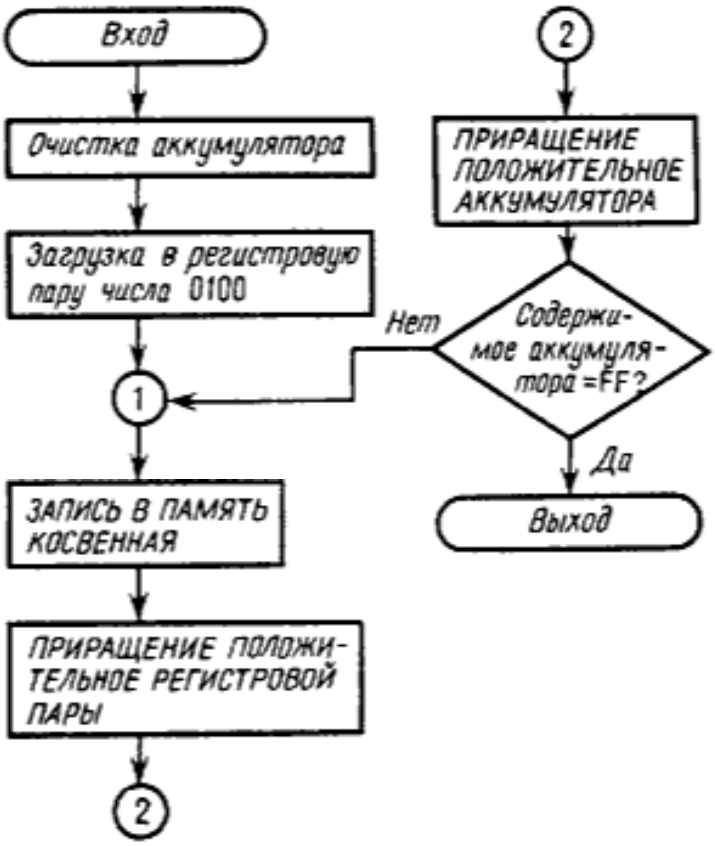


Рис. 7.24. Блок-схема алгоритма к ответу на вопрос п. 10 заданий для самопроверки.

11. При подаче сигнала «Чтение устройства ввода-вывода» обращения к памяти не происходит.
12. Выполнение программы продолжалось бы до записи данных в область FFFF. В результате очередного положительного приращения счетчика его содержимое приняло бы значение 0000 и запись данных была бы продолжена в области памяти, занятые программой. Как только данные были бы записаны в область 0003, произошел бы останов и ее выполнение прекратилось

Глава 8. Арифметические команды

Эта глава знакомит с некоторыми наиболее важными командами микропроцессора. С помощью этих команд производится совместная обработка двух двоичных чисел по арифметическим правилам. Читатель узнает, каким образом применяются эти команды, необходимые для выполнения операций двоичной арифметики. Как будет показано ниже, микропроцессор фактически выполняет все эти арифметические операции за счет использования различными путями команды СЛОЖЕНИЕ.

Сначала будет рассмотрена простая команда СЛОЖЕНИЕ – ее выполнение при различных способах адресации. Затем показано, как используется команда СЛОЖЕНИЕ С ПЕРЕНОСОМ и каким образом она служит для выполнения арифметических операций с повышенной точностью. Далее дается представление о команде ВЫЧИТАНИЕ. Эта команда не выполняет никаких действий, которые нельзя было бы реализовать с помощью команд СЛОЖЕНИЕ и ФОРМИРОВАНИЕ ДОПОЛНИТЕЛЬНОГО КОДА. Объясняется, почему команда ВЫЧИТАНИЕ выполняется быстрее. Для выполнения вычитания с повышенной точностью будет, кроме того, использована команда ВЫЧИТАНИЕ С ПЕРЕНОСОМ.

В заключение рассматривается команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ, показано, каким образом с ее помощью осуществляется быстрое сложение двоично-десятичных чисел.

8.1. Команда СЛОЖЕНИЕ

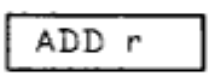
Какими бы мощными устройствами мы ни считали микропроцессоры, многие из них в действительности могут выполнять только одну команду арифметической обработки – СЛОЖЕНИЕ. Способность микропроцессора выполнять разнообразные арифметические действия обусловлена тем обстоятельством, что сложение он может выполнять различными способами, кроме того, очень быстро.

Микропроцессор может, например, сложить одно двоичное число с дополнительным кодом другого двоичного числа. Тем самым реализуется операция вычитания. Сложение может быть повторено множество раз, а многократное прибавление есть не что иное, как операция умножения. Повторяющееся вычитание представляет собой операцию деления. В одной из предшествующих глав мы узнали, как выполняются арифметические действия над двоичными числами. В данном разделе будет показано, как микропроцессор выполняет команду СЛОЖЕНИЕ.

Сейчас мы рассмотрим четыре различные команды сложения. Единственное различие между ними состоит в том, из каких источников берутся данные для их выполнения.

СЛОЖЕНИЕ С РЕГИСТРОМ

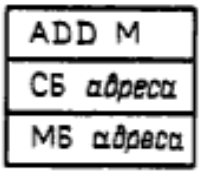
ADD r $A + r \rightarrow A$



По команде СЛОЖЕНИЕ С РЕГИСТРОМ содержимое регистра r прибавляется к содержимому аккумулятора (регистра A). Результат (сумма) помещается в аккумулятор. Исходное содержимое аккумулятора теряется. Это 1-байтовая команда, для выполнения которой затрачиваются два микроцикла процессора. Если результат выполнения команды содержит 1 в старшем разряде или равен нулю либо если возникает перенос из старшего разряда результата, то соответствующие разряды регистра состояния устанавливаются в 1.

СЛОЖЕНИЕ С ПАМЯТЬЮ ПРЯМОЕ

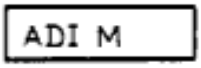
ADD M,Адрес $A + M \rightarrow A$



По команде СЛОЖЕНИЕ С ПАМЯТЬЮ ПРЯМОЕ содержимое области памяти, адрес которой указан в команде, прибавляется к содержимому аккумулятора (регистра A). Второй и третий байты команды составляют адрес области памяти, в которой находится одно из слагаемых. Результат (сумма) помещается в аккумулятор. Исходное содержимое аккумулятора теряется. Это 3-байтовая команда, для выполнения которой затрачиваются четыре микроцикла процессора. Если результат выполнения команды содержит 1 в старшем разряде или равен нулю либо если возникает перенос из старшего разряда результата, то соответствующие разряды регистра состояния устанавливаются в 1.

СЛОЖЕНИЕ С ПАМЯТЬЮ КОСВЕННОЕ

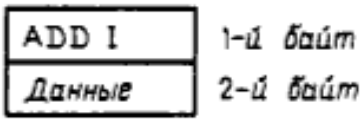
ADI M $A + M \rightarrow A$



При выполнении команды СЛОЖЕНИЕ С ПАМЯТЬЮ КОСВЕННОЕ содержимое области памяти прибавляется к содержимому аккумулятора (регистра A). На область, в которой находится слагаемое, указывает содержимое регистровой пары BC. (Для того чтобы можно было использовать данную команду, регистровая пара BC должна быть предварительно загружена.) Результат (сумма) помещается в аккумулятор. Исходное содержимое аккумулятора теряется. Это 1-байтовая команда, выполняемая за три микроцикла. Если результат выполнения команды содержит 1 в старшем разряде или равен нулю либо если возникает перенос из старшего разряда результата, то соответствующие разряды регистра состояния устанавливаются в 1.

СЛОЖЕНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ

ADD I,Данные $A + Данные \rightarrow A$



При реализации команды СЛОЖЕНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ содержимое второго байта команды прибавляется к содержимому аккумулятора. Результат помещается в аккумулятор. Исходное содержимое аккумулятора теряется. Это – 2-байтовая команда, выполнение которой занимает три микроцикла. Если результат выполнения команды содержит 1 в старшем разряде или равен нулю либо если возникает перенос из старшего разряда результата, то соответствующие разряды регистра состояния устанавливаются в 1.

Теперь мы можем рассмотреть пример того, как может быть использована команда СЛОЖЕНИЕ. На рис. 8.1 приведена блок-схема программы, содержащей эту команду. Выполнение программы иллюстрируется рис. 8.2–8.6. В данной программе используется также команда пересылки данных ЗАГРУЗКА НЕПОСРЕДСТВЕННАЯ. Она служит для размещения в аккумуляторе десятичного числа 12. С помощью команды СЛОЖЕНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ к этому числу 12 прибавляется число 41. Результат (53) остается в аккумуляторе. По

завершении сложения происходит останов процессора.

Хотя на рисунках показаны шестнадцатеричные числа (0С, 29, 35), в действительности микропроцессор обрабатывает

Рис. 8.1. Сложение двух чисел с помощью простой команды СЛОЖЕНИЕ.

двоичные числа. Таким образом, выполняется следующее действие:

Двоичные числа	Десятичные числа	Шестнадцатеричные числа
0000 1100	12	0С
+ 0010 1001	+ 41	+ 29
0011 0101	53	35

На рис. 8.2 показано состояние устройств микро-ЭВМ после выборки команды LDA. Это 2-байтовая команда, реализация которой занимает три микроцикла. В начале работы счетчику команд присваивается значение 0000; при этом он указывает на первую команду программы.

В процессе цикла выборки команды содержимое счетчика команд загружается в регистр адреса памяти. Дешифратор адреса памяти декодирует данные, поступающие по адресной шине. После поступления импульсного сигнала «Чтение памяти» команда LDA из области памяти 0000 помещается на шину данных микро-ЭВМ, а с этой шины записывается в регистр команд микропроцессора.

Во время цикла выполнения (рис. 8.3) дважды производится приращение содержимого счетчика команд. Теперь он указывает на область 0002 – место, где хранится в памяти очередная команда – СЛОЖЕНИЕ. Регистр адреса памяти задает второй байт первой команды, содержащий данные 0С. Дешифратор адреса памяти декодирует адрес данного байта, и по получении импульсного сигнала «Чтение памяти» данные (число 0С) поступают на шину данных микро-ЭВМ. При выполнении команды LDA данные (0С) загружаются в аккумулятор. Можно отметить, что в результате выполнения команды LDA все разряды регистра состояния устанавливаются в 0. Это означает, что значение данных не равно 0, старший их разряд не равен 1, и переноса из старшего разряда при выполнении операции не произошло.

На рис. 8.4 представлена выборка команды СЛОЖЕНИЕ. Во время этого цикла содержимое счетчика команд загружается в регистр адреса памяти. Дешифратор адреса памяти декодирует адрес. По получении сигнала «Чтение памяти» на шину данных подается код команды СЛОЖЕНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ. Эта команда загружается в регистр команд микропроцессора.

В ходе цикла выполнения данной команды (рис. 8.5) дважды осуществляется приращение содержимого счетчика команд. Он указывает теперь на область с адресом 0004, в которой хранится команда ОСТАНОВ. Содержимое аккумулятора пересылается в буферный регистр аккумулятора. В то же время дешифратор адреса памяти указывает на данные, содержащиеся во втором байте команды. Дешифратор адреса памяти указывает на область с адресом 0003. После поступления сигнала «Чтение памяти» данные (число 29) направляются на шину данных микро-ЭВМ, откуда они загружаются в буферный регистр АЛУ.

Затем АЛУ складывает содержимое буферного регистра АЛУ и аккумулятора. Результат (число 35) помещается в аккумулятор.

Разрядам регистра состояния, соответствующим равенству результата нулю, наличию у него знака минус и появлению переноса из старшего разряда, присваиваются необходимые значения. В данном случае все они устанавливаются в 0.

На рис. 8.6 можно видеть результаты выполнения команды СЛОЖЕНИЕ С НЕ-

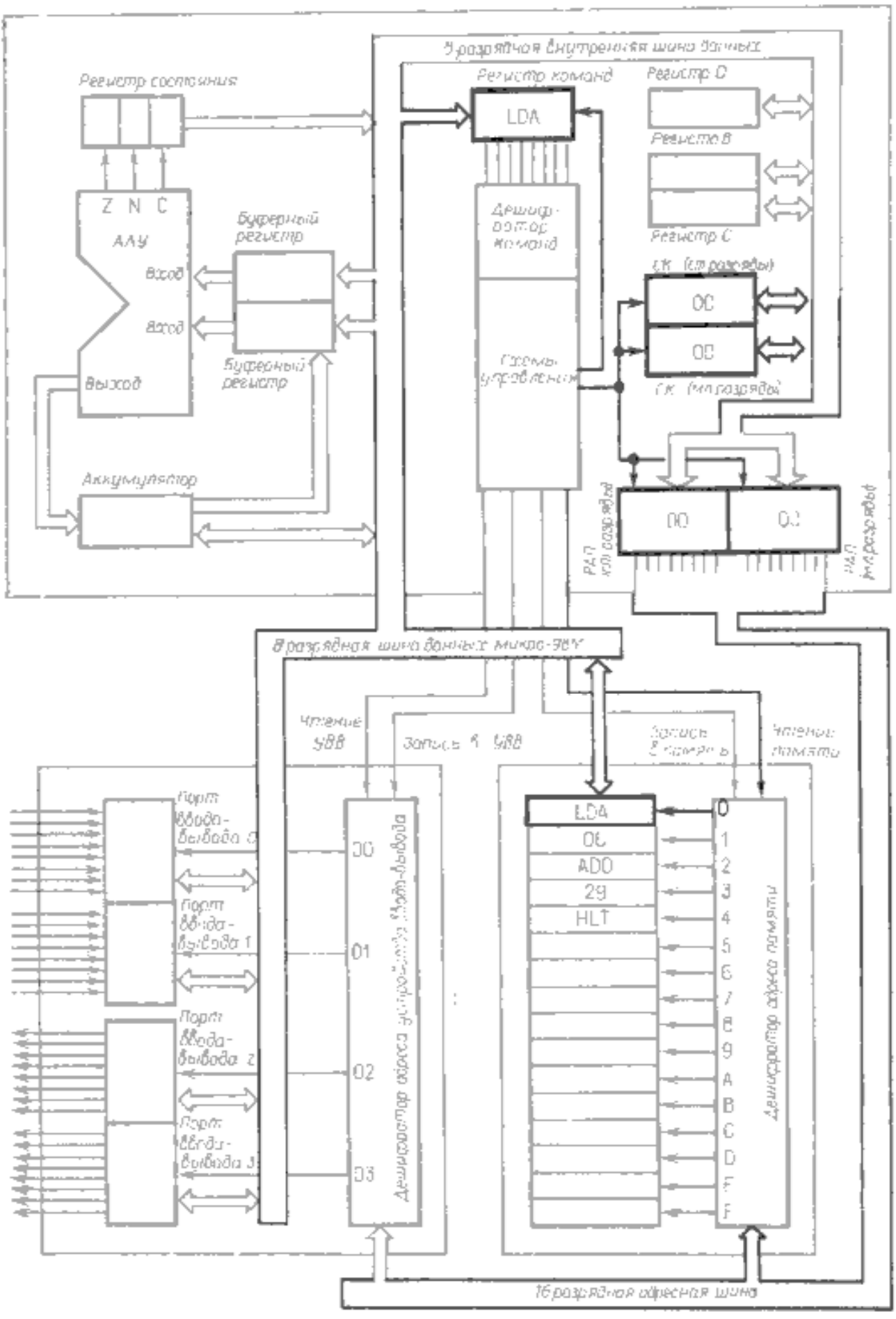


Рис. 8.2. Выборка команды ЗАГРУЗКА АККУМУЛЯТОРА (LDA).

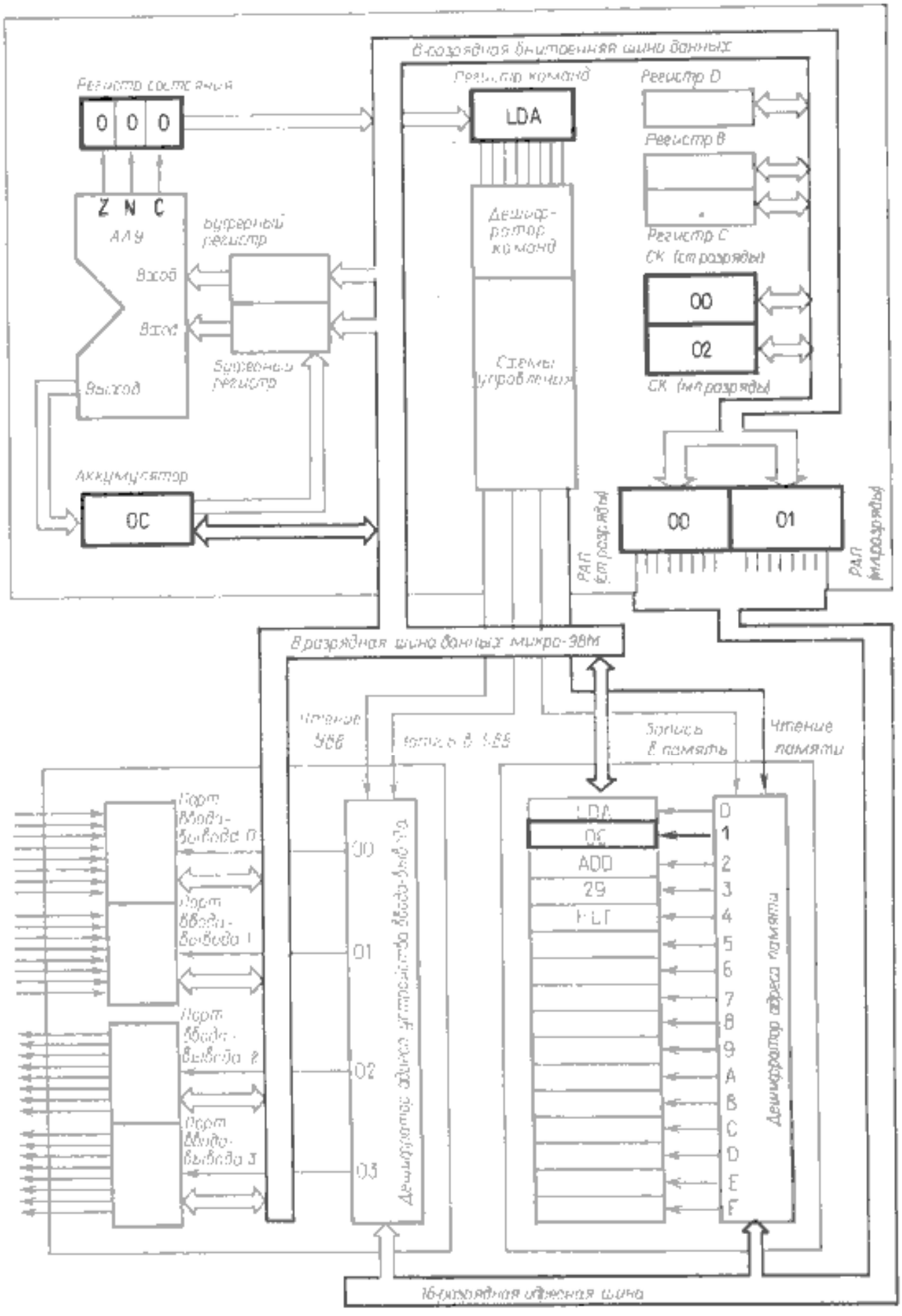


Рис. 8.3. Выполнение команды LDA. (Данные, содержащиеся во втором байте команды LDA, загружаются в аккумулятор, и устанавливаются в 0 все разряды регистра состояния.)

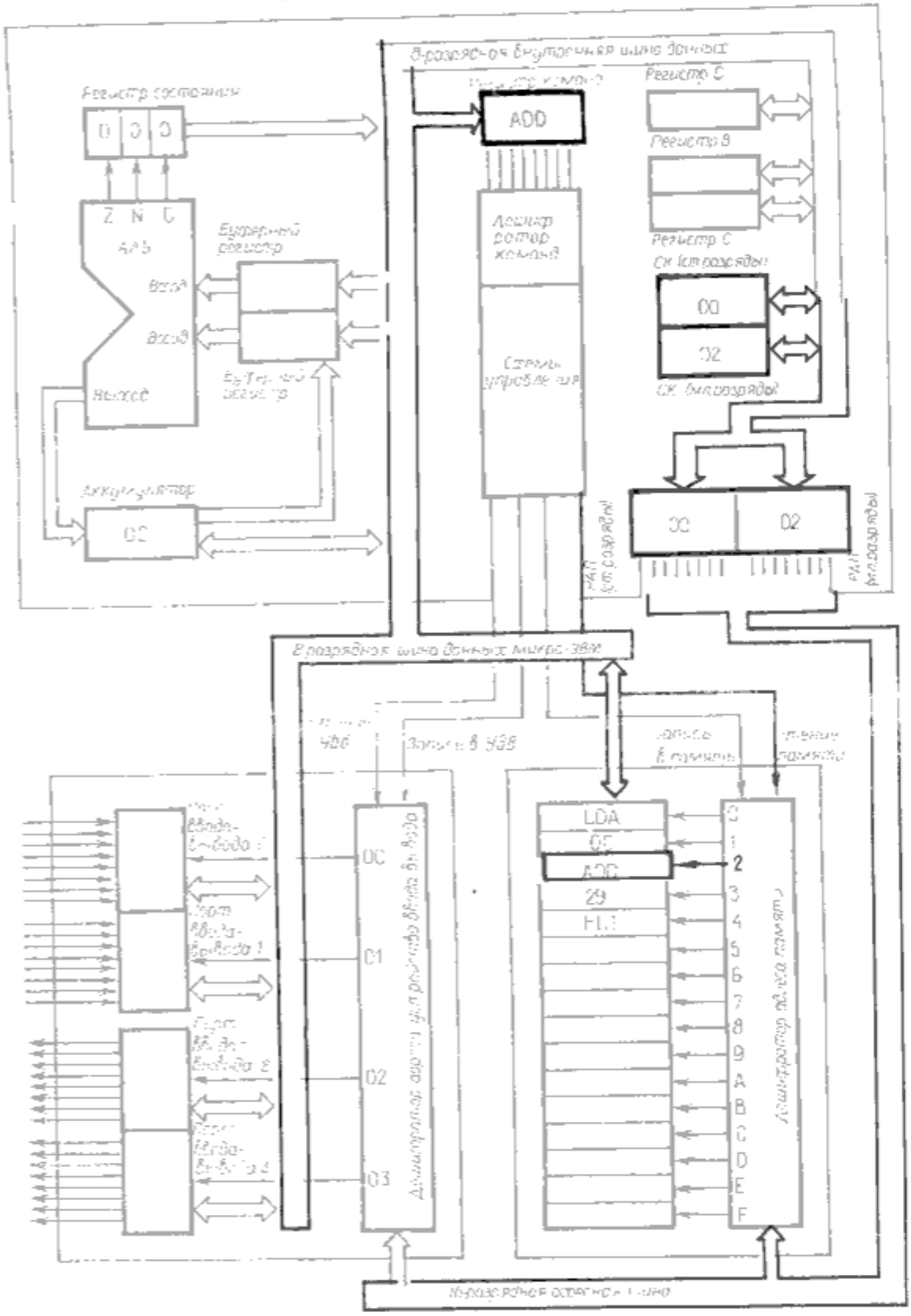


Рис. 8.4. Выборка команды СЛОЖЕНИЕ.

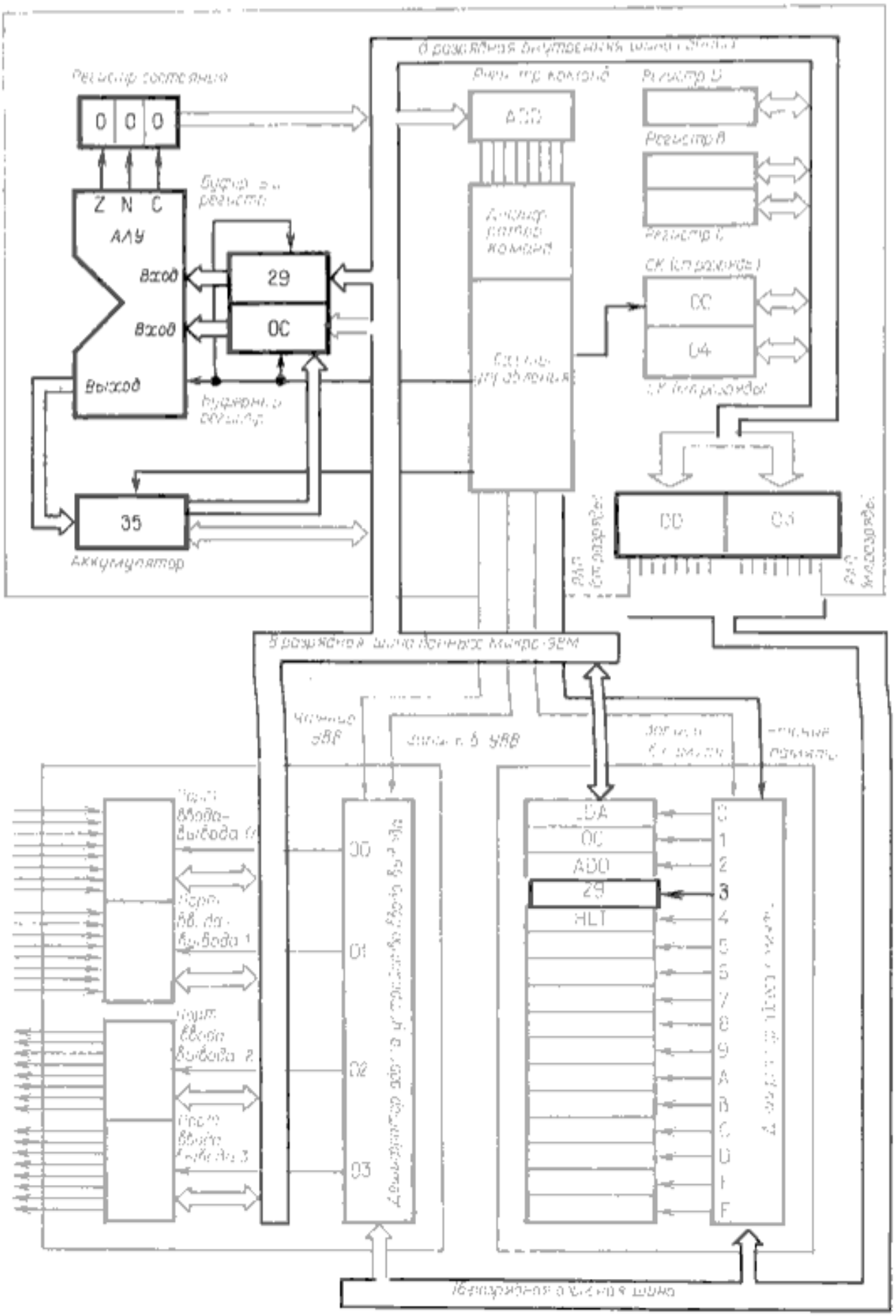


Рис. 8.5. Выполнение команды СЛОЖЕНИЕ. Содержимое аккумулятора помещается в буферный регистр аккумулятора. Затем АЛУ складывает содержимое буферного регистра аккумулятора с данными, содержавшимися в команде СЛОЖЕНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ, которые находятся в буферном регистре.

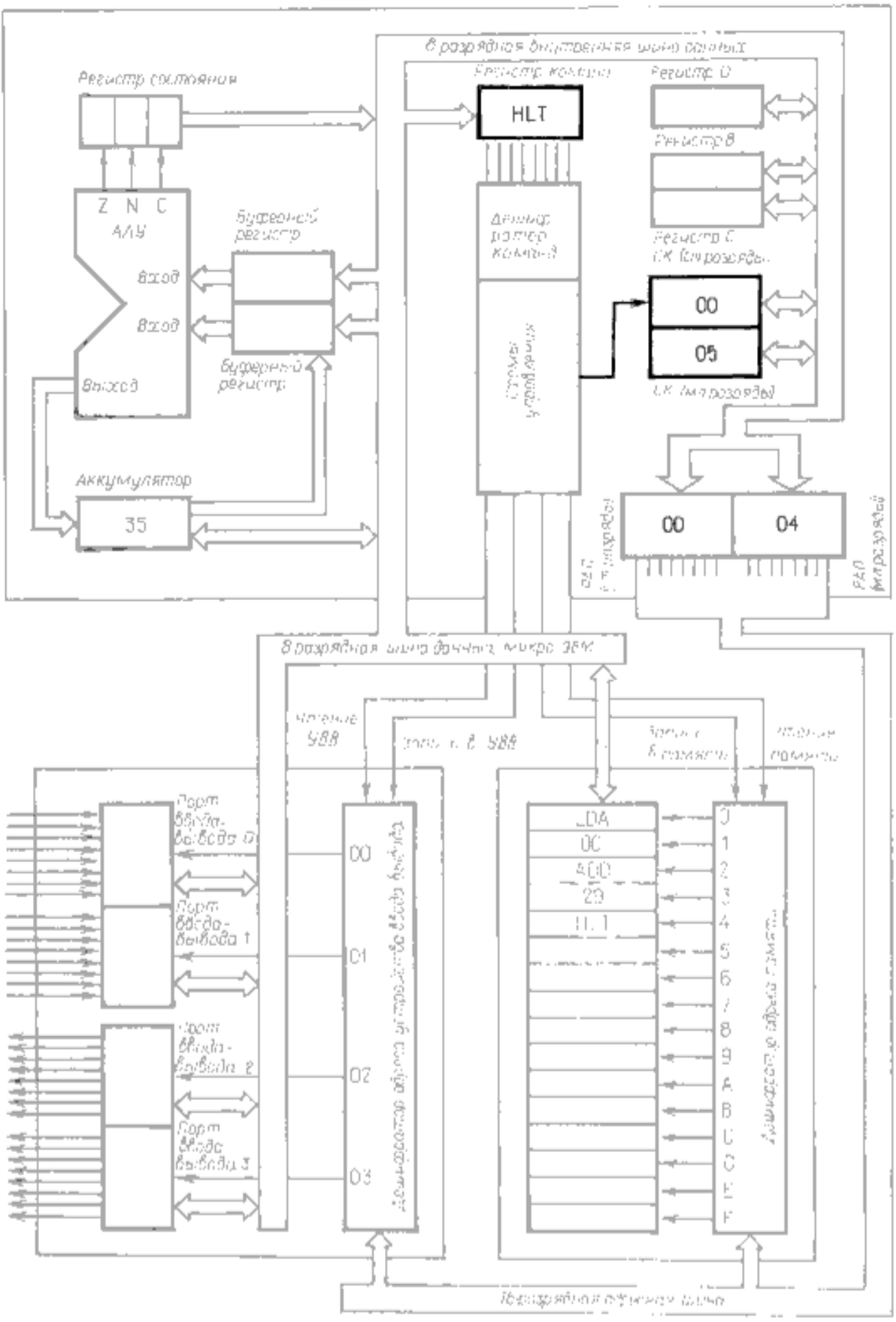


Рис. 8.6. Выполнение команды ОСТАНОВ (прекращение работы микропроцессора).

ПОСРЕДСТВЕННЫМИ ДАННЫМИ. Во время выборки содержимое счетчика команд было загружено в регистр адреса памяти. Дешифратор адреса памяти декодировал адрес, поступивший по адресной шине. При поступлении сигнала «Чтение памяти» на шину данных микро-ЭВМ была подана команда ОСТАНОВ. Затем эта команда была загружена в регистр команд.

Во время выполнения команды ОСТАНОВ производится положительное приращение счетчика команд, который указывает на область 0005, содержащую следующую команду. Однако устройство управления декодирует команду ОСТАНОВ и прекращает работу микропроцессора.

Программа выполнена полностью. В аккумуляторе находятся данные (число 35), являющиеся результатом сложения чисел 0С и 29.

Может показаться, что для сложения двух чисел приходится производить большое число действий. Конечно, для человека такое сложение было бы довольно утомительным занятием, но микропроцессор делает это так быстро! Предположим, что каждый микроцикл процессора длится 1 мкс. Прикинем, за какое время он выполнит полностью операцию сложения:

Команда	Количество микроциклов (1 мкс/микроцикл)
LDA r	3
ADD I	+ 3
	6

Выполнение всей программы занимает шесть микроциклов, что составит 6 мкс при длительности каждого микроцикла 1 мкс. Это означает, что за одну секунду микропроцессор может выполнить 166 667 таких программ. Если же время микроцикла равно 200 нс, то за секунду таких программ может быть выполнено 833 333.

Задания для самопроверки

1. Часто после выполнения команды СЛОЖЕНИЕ ее результат помещается в память. Изобразите блок-схему необходимой для этого программы и объясните, как выполняется команда ЗАПИСЬ В ПАМЯТЬ ПРЯМАЯ. Как скажется на времени выполнения программы наличие в ней этой команды при длительности микроцикла процессора 1 мкс?

2. Предположим, что вам необходимо многократно использовать программу, рассмотренную в качестве примера. Необходимо складывать содержимое областей памяти с адресами 0008 и 0009, а результат записывать в область с адресом 000А. Какие команды для этого понадобятся? Чем соответствующая программа будет отличаться от рассмотренной в примере?

3. Измените программу, рассмотренную в примере, так, чтобы она содержала команду СЛОЖЕНИЕ С РЕГИСТРОМ. Объясните отличия полученной таким образом программы от исходной, описанной в примере, и от разработанной при ответе на п. 2 заданий.

8.2. Команда СЛОЖЕНИЕ С ПЕРЕНОСОМ

Кроме четырех команд СЛОЖЕНИЕ, которые мы рассмотрели, существуют еще четыре разновидности этих команд. Они очень похожи на команды первых четырех типов и отличаются тем, что при их выполнении в сложении участвует содержимое разряда переноса регистра состояния. Таким образом, содержимое области памяти прибавляется к содержимому аккумулятора, а затем к полученной сумме прибавляется значение разряда переноса из регистра состояния. Рассмотрим пример такого сложения:

Двоичное представление	Источник данных	Десятичное представление
1	Перенос	1
+ 0001 1001	+ Первое слагаемое	+ 25
+ 0000 1110	+ Второе слагаемое	+ 14
0010 1000	Сумма	40

В данном случае выполняется сложение десятичных чисел 25 и 14. Если бы в сложении участвовали только эти числа, результат был равен 39. Однако к ним прибавляется еще и третье число – это число 1. Оно представляет собой перенос, возникший при выполнении предыдущей арифме-

тической операции. Итак, 1 + 25 + 14 = 40. Команда СЛОЖЕНИЕ С ПЕРЕНОСОМ представляет собой единственное средство, с помощью которого микропроцессор может сложить сразу же три числа. Во всех других случаях, когда требуется сложить три числа, необходимо сначала использовать одну команду СЛОЖЕНИЕ, чтобы сложить два первых числа, а затем с помощью еще одной команды СЛОЖЕНИЕ прибавить третье число к сумме двух первых чисел.

Команда СЛОЖЕНИЕ С ПЕРЕНОСОМ дает возможность производить сложение двух чисел вместе с переносом, полученным в предыдущей операции. Разумеется, при каждом выполнении команды СЛОЖЕНИЕ С ПЕРЕНОСОМ соответствующим образом устанавливаются в регистре состояния разряды, указывающие на равенство результата нулю, наличие у него знака минус и наличие переноса из старшего разряда. Поэтому исходное содержимое и аккумулятора, и регистра состояния утрачивается, как и при выполнении других команд СЛОЖЕНИЕ.

К числу команд СЛОЖЕНИЕ С ПЕРЕНОСОМ относятся следующие команды:

СЛОЖЕНИЕ С РЕГИСТРОМ И ПЕРЕНОСОМ

ACD r A + r + C → A

ACD r

1 байт, два цикла

СЛОЖЕНИЕ С ПАМЯТЬЮ И ПЕРЕНОСОМ ПРЯМОЕ

ACD M,Адрес A + M + C → A

AND M
СБ *адреса*
МБ *адреса*

3 байт, четыре цикла

СЛОЖЕНИЕ С ПАМЯТЬЮ И ПЕРЕНОСОМ КОСВЕННОЕ

ACI M M + A + C → A

ACI M

1 байт, два цикла

СЛОЖЕНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ И ПЕРЕНОСОМ

ACD I,Данные Данные + A + C → A

AND I
Данные

2 байт, три цикла

Обратимся теперь к примеру программы, в которой используется СЛОЖЕНИЕ С ПЕРЕНОСОМ. При этом мы не только рассмотрим, как выполняется команда СЛОЖЕНИЕ С ПЕРЕНОСОМ, но выясним также, каким образом микропроцессор выполняет арифметические операции с повышенной точностью. Ранее мы узнали, что арифметические операции с повышенной точностью используются, когда одного слова данных оказывается недостаточно для представления промежуточного или окончательного результата вычислений.

Например, с помощью одного слова 8-разрядного микропроцессора можно представить только числа, лежащие в диапазоне от 0 до 255. Для представления чисел, больших по величине чем число 255, необходимо использовать больше чем одно слово данных. Понятие арифметики повышенной точности означает использование для представления одного числа размера данных более чем одно слово.

При работе с 8-разрядным микропроцессором, прибегая к двойному формату, т. е. используя два слова для изображения одного числа, можно представить числа от 0 до 65 535. Применяя тройной формат, оказывается возможным представить числа от 0 до 16 777 215. Одинарный, двойной и тройной форматы 8-разрядного слова показаны на рис. 8.7.

При сложении чисел двойного формата сначала складываются их младшие байты. Это делается с помощью обычной команды СЛОЖЕНИЕ, при выполнении которой не используется значение переноса. При этом, конечно, может образоваться перенос из старшего разряда младшего байта суммы. Затем посредством команды

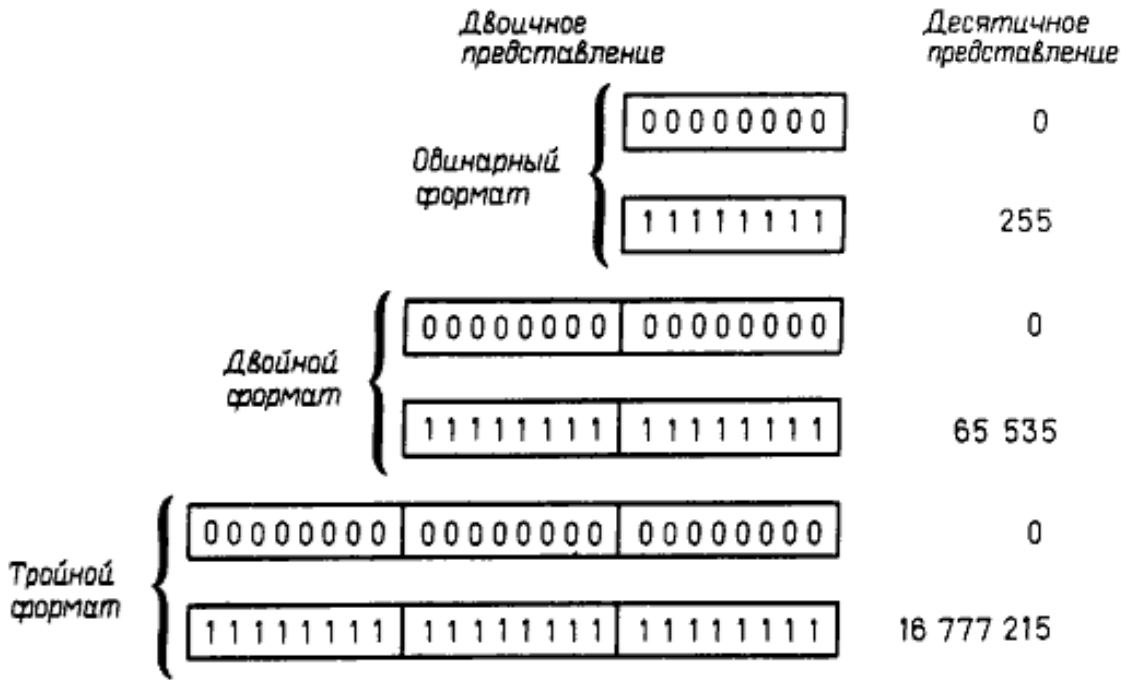


Рис. 8.7. Представление данных в одинарном, двойном и тройном форматах в 8-разрядной вычислительной системе. Точность представления выражается соответственно величинами 1/255, 1/65635 и 1/16777215.

СЛОЖЕНИЕ С ПЕРЕНОСОМ производится сложение старших байтов чисел. Такая команда используется в том случае, если при сложении младших байтов выработан перенос. Если перенос имеет место, то он участвует в выполнении команды СЛОЖЕНИЕ С ПЕРЕНОСОМ; если перенос из младшего байта не был сформирован, на вход переноса подается 0.

Когда числа представлены в тройном формате, необходимо реализовать еще одну команду СЛОЖЕНИЕ С ПЕРЕНОСОМ. При выполнении этой, третьей по счету команды складываются старшие байты двух чисел тройного формата.

Перенос, вырабатываемый в результате выполнения последней операции, входящей в последовательность действий по арифметической обработке повышенной точности, необходимо либо сохранить, либо отбросить. В большинстве случаев предполагается, что при выполнении последней операции перенос не образуется, т. е. применен формат, достаточный для размещения наибольшего возможного результата вместе с его битом знака.

Рассмотрим пример, когда необходимо применение двойного формата. На рис. 8.8 показано сложение двух чисел. Так как результат сложения не превышает число 65 535, эта арифметическая операция может быть выполнена в двойном формате. Двоичные числа представлены без знака. Об этом можно судить по наличию 1 в старшем байте результата. Обратите внимание, что результат представляет собой число 63 580, а не – 30 812.

Как показывает пример, слагаемые и сумма разбиваются на старшие и младшие байты. Для суммирования младших байтов слагаемых используется простое двоичное сложение. С этой целью применяется команда СЛОЖЕНИЕ, в результате выполнения которой формируется младший байт суммы, а также вырабатывается перенос.

По окончании этой операции разряд переноса в регистре состояния установлен в 1. Для суммирования старших байтов слагаемых необходимо использовать команду СЛОЖЕНИЕ С ПЕРЕНОСОМ. При ее выполнении единичное значение бита переноса участвует в вычислении. Как видно на рис. 8.8, значение первых четырех битов старшего байта результата изменяется именно благодаря наличию этого переноса.

Блок-схема программы изображена на

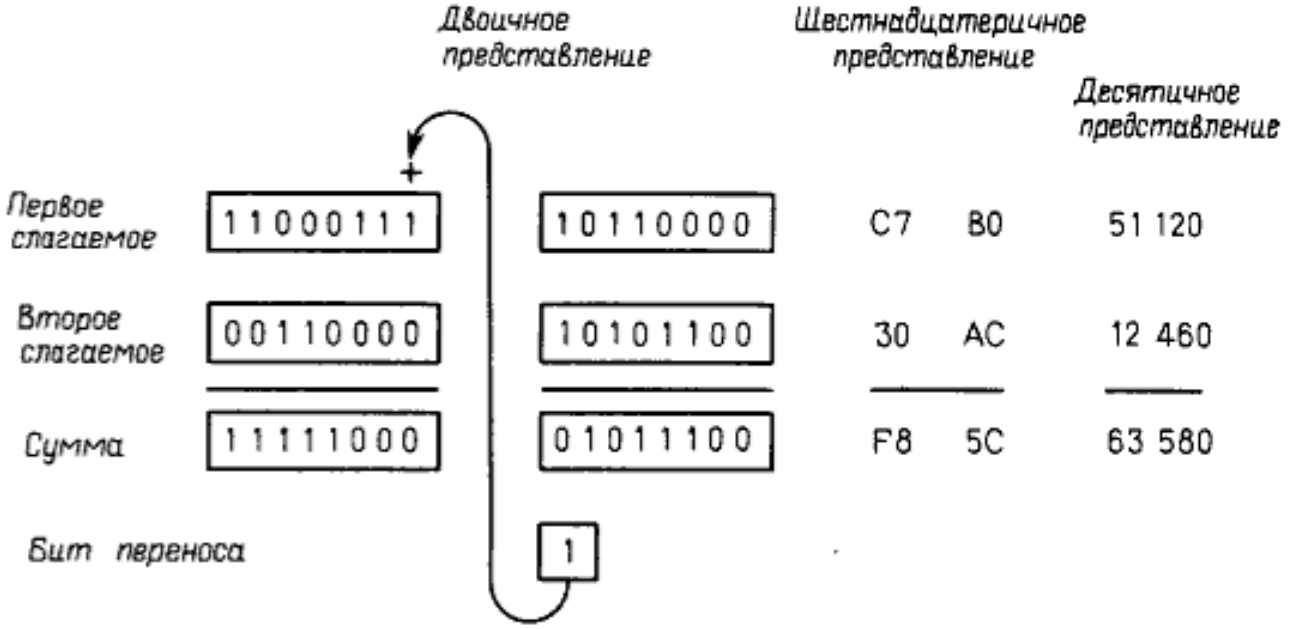


Рис. 8.8. Сложение чисел двойного формата, в процессе которого имеет место перенос из младшего байта в старший. В сложении участвуют два 16-разрядных двоичных числа без знака.

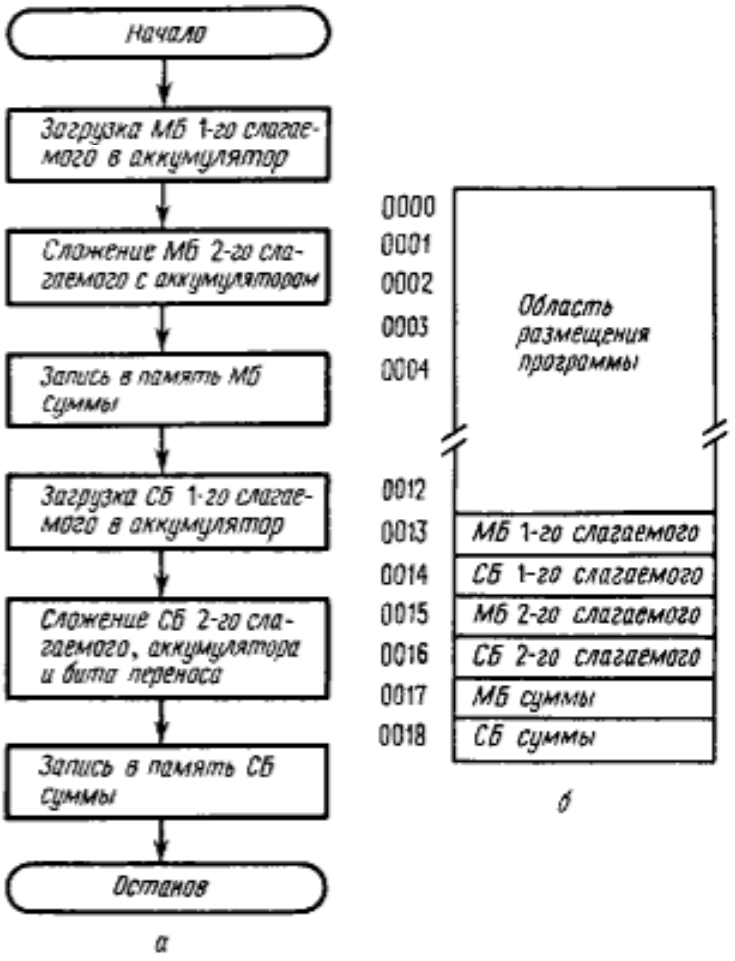


Рис. 8.9. Блок-схема (*а*) и схема распределения памяти (*б*) для программы сложения чисел двойного формата. Из схемы распределения памяти следует, что 6 байт адресного пространства отведено для размещения участвующих в сложении чисел двойного формата.

рис. 8.9, *а*. Вначале складываются младшие байты, а затем старшие. В сложении старших байтов участвует бит переноса.

Рис. 8.9, *б* – это схема распределения памяти. Она показывает, как используется память микропроцессорной системы при выполнении данной программы. Программа размещается в областях 0000...0012. В областях 0013 и 0014 содержатся соответственно младший и старший байты первого слагаемого. Младший и старший байты второго слагаемого хранятся в областях 0015 и 0016. Младший и старший байты суммы записываются в области 0017 и 0018.

Рис. 8.10 представляет собой листинг программы, предназначенной для выполнения рассмотренного выше сложения. Данные, приведенные на этом рисунке, сгруппированы в три колонки. В левой из них перечислены шестнадцатеричные адреса памяти. В средней колонке показано содержимое приведенных областей памяти. Это содержимое состоит из мнемонических обозначений команд и шестанадцатеричных чисел, являющихся адресами памяти. Программа начинается с области 0000. Память данных находится в последних шести областях. (Можно сопоставить программу, представленную на рис. 8.10, со схемой выполнения сложения, приведенной на рис. 8.8.) В правой колонке рис. 8.10 имеются пояснения, с помощью которых

Адрес области памяти	Содержимое области памяти	Комментарий
0000	LDD A	ЗАГРУЗКА АККУМУЛЯТОРА ПРЯМАЯ
0001	00	Адрес младшего байта
0002	13	первого слагаемого
0003	ADD A	СЛОЖЕНИЕ С АККУМУЛЯТОРОМ ПРЯМОЕ
0004	00	Адрес младшего байта
0005	15	второго слагаемого
0006	STA A	ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ ПРЯМАЯ
0007	00	Адрес младшего байта суммы
0008	17	
0009	LDD A	ЗАГРУЗКА АККУМУЛЯТОРА ПРЯМАЯ
000A	00	Адрес старшего байта
000B	14	первого слагаемого
000C	ACD A	СЛОЖЕНИЕ С ПЕРЕНОСОМ ПРЯМОЕ
000D	00	Адрес старшего байта
000E	16	второго слагаемого
000F	STA A	ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ ПРЯМАЯ
0010	00	Адрес старшего байта суммы
0011	18	
0012	HLT	ОСТАНОВ
0013	B0	Младший байт Первое слагаемое
0014	C7	Старший байт (51 120)
0015	AC	Младший байт Второе слагаемое
0016	30	Старший байт (12 460)
0017	5C	Младший байт Сумма
0018	F8	Старший байт (63 580)

Рис. 8.10. Листинг программы сложения чисел двойного формата.

легче разобраться в программе. Программы, написанные от руки, часто представляют в подобной форме.

Рассматривая программу, легко заметить, что в ней часто используется команда СЛОЖЕНИЕ ПРЯМОЕ. Применение этой команды обусловлено тем обстоятельством, что области памяти, в которых размещаются слагаемые и сумма двойного формата, не входят в состав команд, осуществляющих сложение. Действительно, программа сложит любые числа, которые окажутся в областях первого и второго слагаемых. Команда ЗАПИСЬ В ПАМЯТЬ служит для указания области, предназначенной для размещения суммы. Если в ходе выполнения программы возникнет перенос при сложении старших байтов, он будет утерян. Другими словами, данная программа может складывать только двоичные числа без знака, сумма которых не превышает 65 535, или двоичные числа со знаком, сумма которых лежит в диапазоне от $-32\,768$ до $+32\,767$.

Проследим выполнение программы с помощью структурной схемы микропроцес-

сорной системы. Каждое из рассматриваемых ниже состояний системы соответствует результатам выполнения очередной команды. Здесь уже не приводятся состояния, имеющие место после каждого микроцикла процессора.

Рис. 8.11 иллюстрирует, что делает первая команда программы. При ее выполнении микро-ЭВМ загружает значение области 0013 в аккумулятор.

В цикле выборки содержимое счетчика команд пересылается в регистр адреса памяти. Дешифратор адреса памяти декодирует адрес. При поступлении импульса «Чтение памяти» значение области 0000 подается на шину данных микро-ЭВМ. Так как происходит цикл выборки, это значение загружается в регистр команд. Устройство управления декодирует команду ЗАГРУЗКА АККУМУЛЯТОРА ПРЯМАЯ.

В ходе цикла выполнения команды содержимое областей памяти 0001 и 0002 загружается в регистр адреса памяти. Регистр адреса памяти указывает на область 0013. Содержимое области 0013 загружается в аккумулятор. Таким образом, в аккумулятор загружен младший байт первого слагаемого.

На рис. 8.12 показаны результаты выполнения второй команды. По команде СЛОЖЕНИЕ ПРЯМОЕ значение области памяти прибавляется к содержимому аккумулятора.

Во время цикла выборки содержимое счетчика команд загружается в регистр адреса памяти. При поступлении сигнала «Чтение памяти» в регистр команд загружается команда СЛОЖЕНИЕ ПРЯМОЕ.

В процессе выполнения второй команды реализуются три независимых действия. Во-первых, осуществляется положительное приращение содержимого счетчика команд таким образом, чтобы он указывал на местоположение в памяти следующей команды, т.е. на область 0006. Во-вторых, устройство управления микропроцессора загружает остальные 2 байт команды в регистр адреса памяти; при этом последний указывает на область 0015. В-третьих, содержимое аккумулятора пересылается в буферный регистр.

Значение области памяти 0015 загружается в буферный регистр АЛУ. Арифметическо-логическое устройство производит сложение чисел, поступающих на два его входа, и полученную сумму помещает в аккумулятор. Как показано на схеме, разряд переноса в регистре состояния установлен в 1, так как при сложении возник перенос из старшего разряда.

Рис. 8.13 демонстрирует, что при реализации команды ЗАПИСЬ В ПАМЯТЬ ПРЯМАЯ содержимое аккумулятора записывается в область памяти с адресом 0017. Теперь в области памяти 0017 находится младший байт суммы. При выполнении данной операции три раза происходит чтение информации из памяти и один раз запись в память.

В цикле выборки данной команды код ее операции загружается в регистр команд. Для выполнения команды старший и младший байты адреса считываются из памяти и помещаются в регистр адреса памяти. Теперь регистр адреса памяти указывает на область 0017. Содержимое аккумулятора поступает на шину данных микро-ЭВМ. Устройство управления выдает сигнал «Запись в память». По этому сигналу данные, находящиеся на шине данных, записываются в область памяти 0017.

На рис. 8.14 показано, как происходит пересылка информации в аккумулятор с помощью команды ЗАГРУЗКА АККУМУЛЯТОРА ПРЯМАЯ. Эта операция протекает так же, как и аналогичная операция, выполнение которой иллюстрируется рис. 8.11. Различие состоит в том, что источником данных здесь служит область 0014, в которой содержится старший байт первого слагаемого.

Рис. 8.15 отражает выполнение команды СЛОЖЕНИЕ С ПЕРЕНОСОМ ПРЯМОЕ. По этой команде старший байт второго слагаемого складывается с битом переноса и содержимым аккумулятора. Результат загружается в аккумулятор. В фазе выборки команды содержимое счетчика команд загружается в регистр адреса памяти. Дешифратор адреса памяти определяет адрес области 000С. При поступлении сигнала «Чтение памяти» значение области памяти с адресом 000С подается на шину данных микро-ЭВМ. В регистр команд микропро-

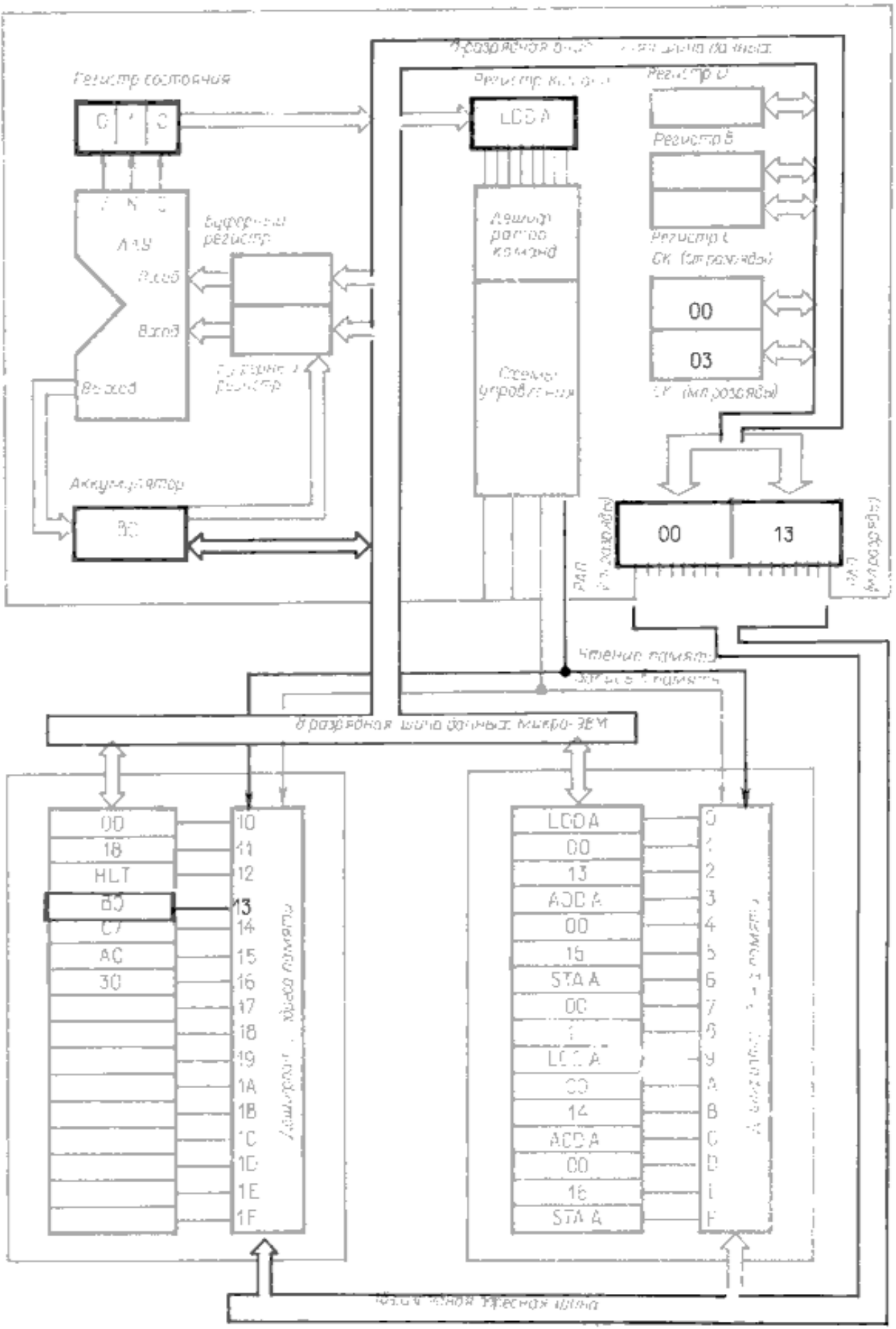


Рис. 8.11. Выполнение команды LDA A,0013. (Пересылка данных из области памяти 0013 в аккумулятор; установка в 1 разряда отрицательного результата в регистре состояния.)

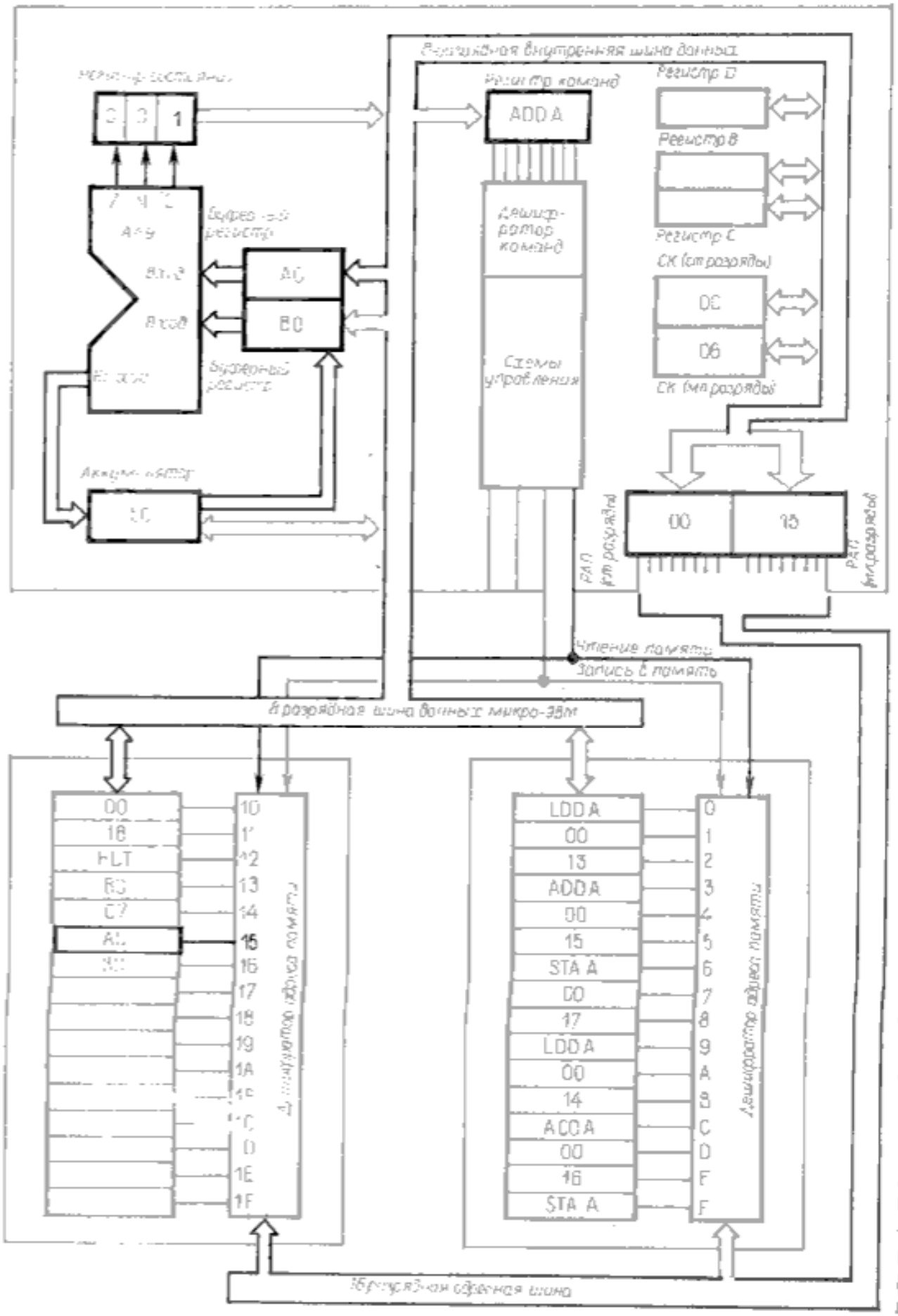


Рис. 8.12. Выполнение команды ACD M,0015. (Прибавление значения области памяти с адресом 0015 к содержимому аккумулятора. Установка в 1 разряда переноса в регистре состояния в связи с возникновением арифметического переноса. Поскольку результат на является отрицательным, разряд, соответствующий наличию знака минус у результата, устанавливается в 0.)

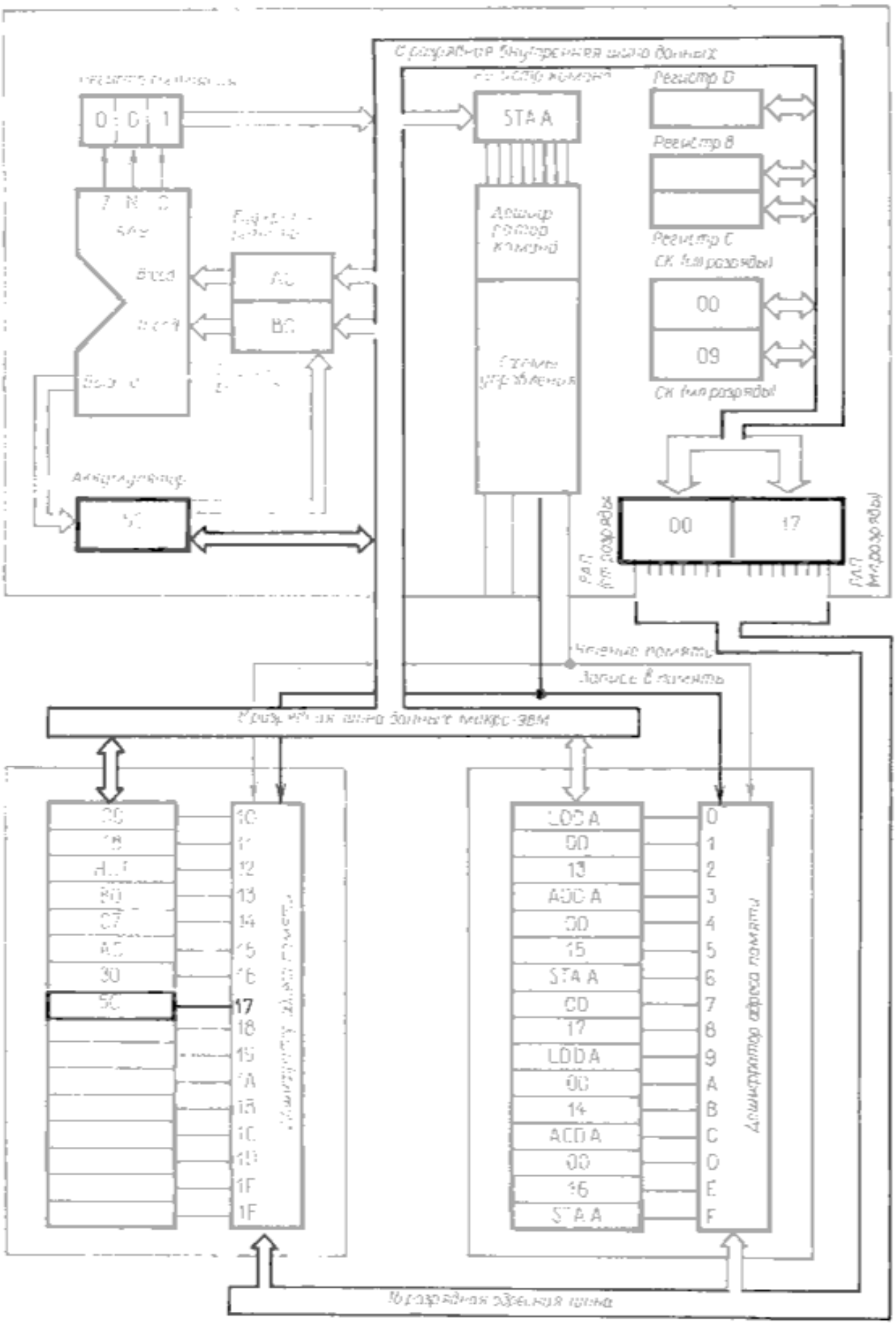


Рис. 8.13. Пересылка содержимого аккумулятора в память. (При выполнении команды ЗАПИСЬ В ПАМЯТЬ ПРЯМАЯ по импульсному сигналу «Запись в память» данные помещаются в область памяти с адресом 0017.)

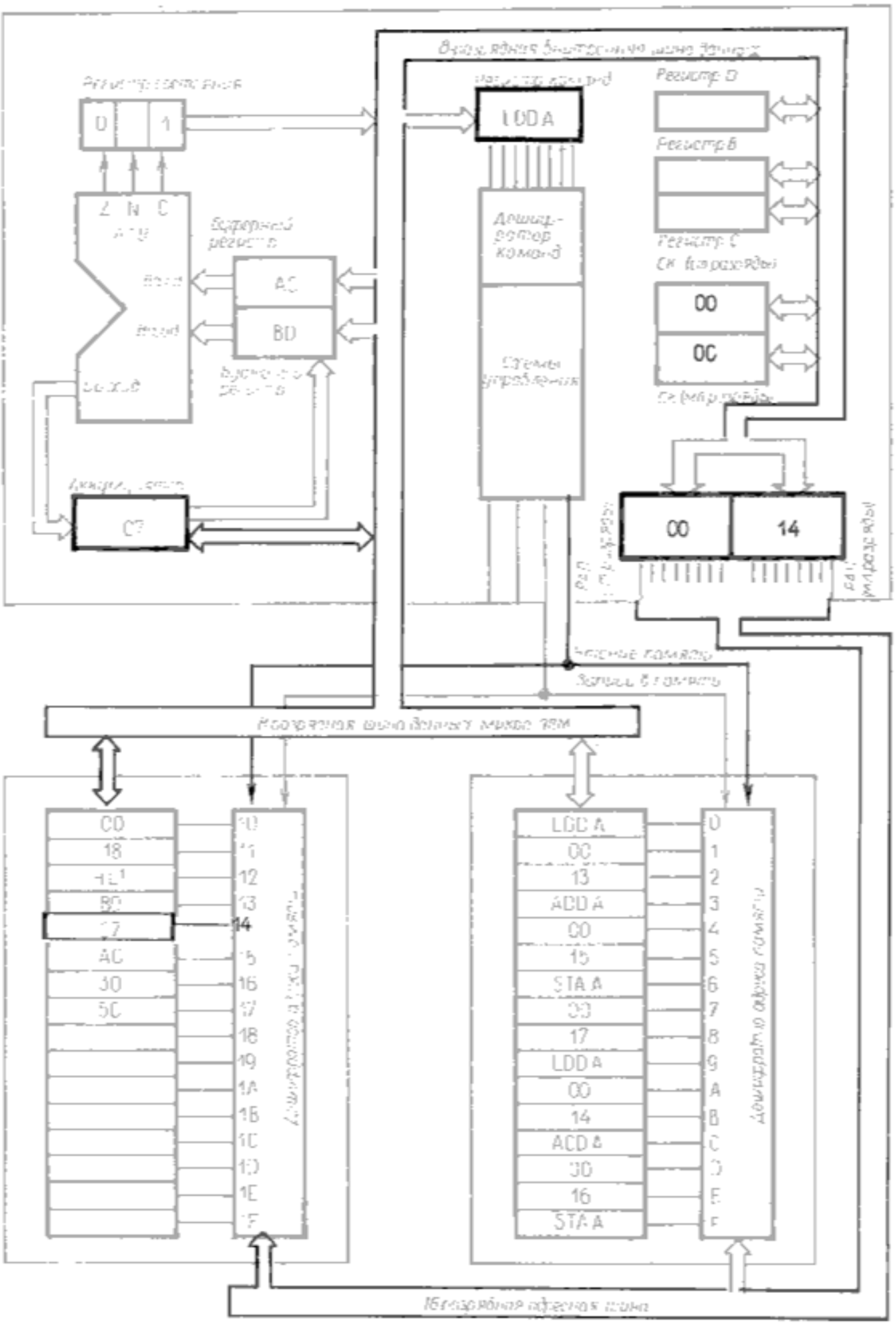


Рис. 8.14. Загрузка в аккумулятор старшего байта первого слагаемого. (При выполнении команды ЗАГРУЗКА АККУМУЛЯТОРА ПРЯМАЯ разряды нулевого результата и отрицательного результата устанавливаются соответствующим образом, а значение разряда переноса остается без изменений. Оно сохраняется таким, каким стало при выполнении предыдущей операции СЛОЖЕНИЕ.)

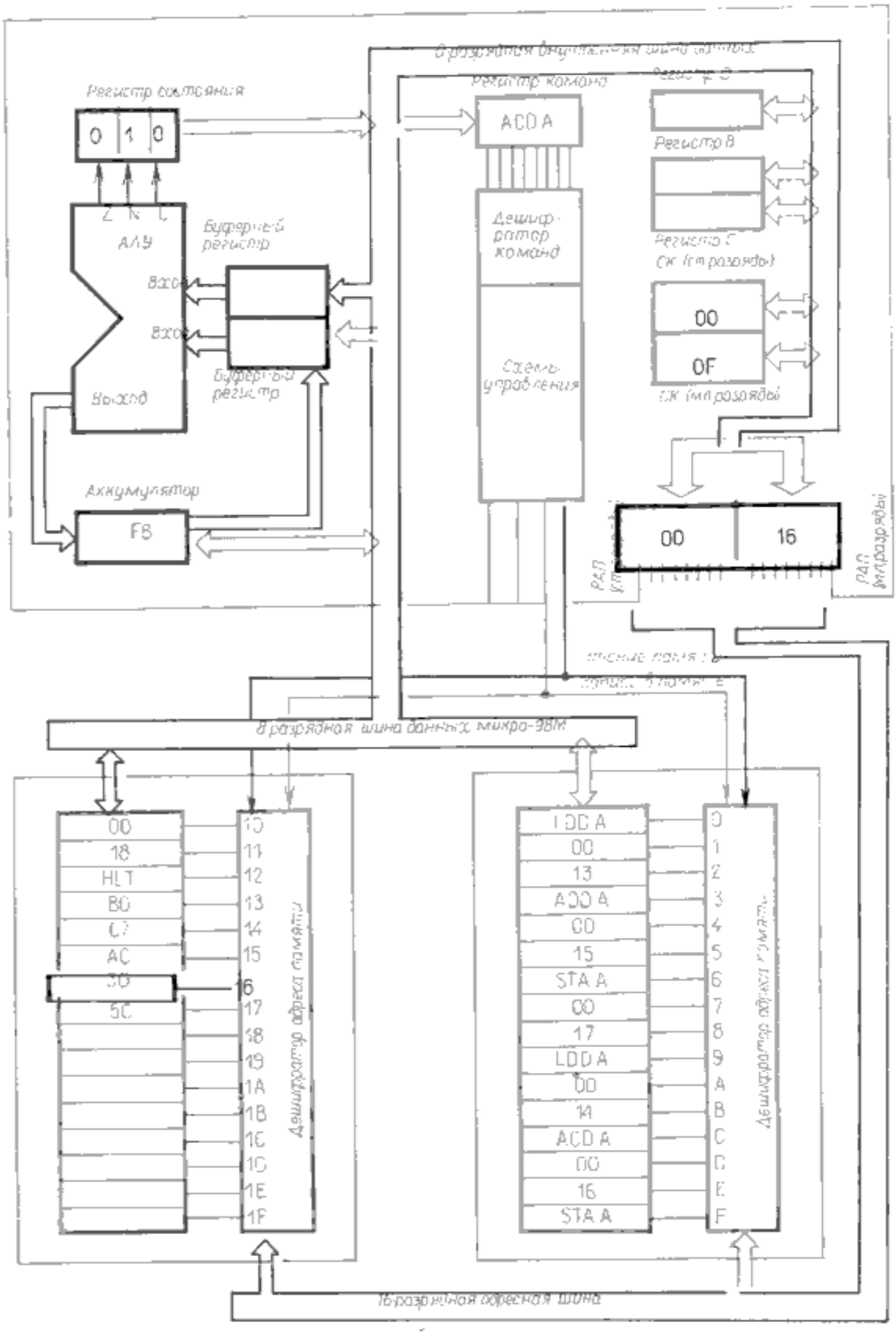


Рис. 8 15. Выполнение команды СЛОЖЕНИЕ С ПЕРЕНОСОМ ПРЯМОЕ. (Значение области памяти с адресом 0016 прибавляется к содержимому аккумулятора, которое было загружено в буферный регистр. В сложении участвует также разряд переноса регистра состояния – в данном случае его значение равно 1. При сложении вырабатывается новое слово состояния, которое отражает наличие логической 1 в старшем разряде аккумулятора)

цессора помещается код операции СЛОЖЕНИЕ С ПЕРЕНОСОМ ПРЯМОЕ.

В фазе выполнения команды осуществляется положительное приращение содержимого счетчика команд, после чего он указывает на область 000F. В регистр адреса памяти загружаются второй и третий байты команды. Теперь этот регистр указывает на область 0016. Содержимое аккумулятора пересылается в буферный регистр. Наконец, содержимое области памяти 0016, бит переноса и содержимое буферного регистра складываются, и полученная сумма загружается в аккумулятор.

По завершении этой команды разряд переноса сбрасывается в 0, так как при выполнении команды СЛОЖЕНИЕ С ПЕРЕНОСОМ перенос не вырабатывается. Однако разряд отрицательного результата в регистре состояния устанавливается в 1, потому что такое значение приобретает в конце операции старший разряд аккумулятора. В аккумуляторе сейчас находится старший байт суммы.

На рис. 8.16 показана запись содержимого аккумулятора в область с адресом 0018. Команда, выполняющая это действие, аналогична команде, которую иллюстрирует рис. 8.13, и отличается лишь тем, что в данном случае в память (область с адресом 0018) записывается старший байт суммы.

На этом сложение двух чисел двойного формата, в результате которого формируется сумма двойного формата, заканчивается. Результат сложения помещается в области памяти с адресами 0017 и 0018 (рис. 8.16). Рассмотрение данного примера показывает, что аналогичные вычисления над числами тройного формата можно реализовать, выполнив еще одну команду сложения с переносом.

Задания для самопроверки

4. Какое из нижеперечисленных действий является назначением команды СЛОЖЕНИЕ С ПЕРЕНОСОМ:

а) выработка в случае необходимости единичного значения бита переноса;

б) установка в 1 разряда переноса в регистре состояния;

в) принятие во внимание при выполнении операции значения переноса, образованного в ходе реализации предыдущей операции;

г) запоминание бита переноса?

5. Что произойдет, если в программе, приведенной на рис. 8.8, команду СЛОЖЕНИЕ, находящуюся в области 0003, заменить на СЛОЖЕНИЕ С ПЕРЕНОСОМ, а команду, размещенную в области 000C,– на команду СЛОЖЕНИЕ?

6. Покажите, какие изменения надо внести в программу, представленную на рис. 8.8, чтобы она выполняла сложение чисел тройного формата.

7. Как сказывается на времени выполнения замена команды СЛОЖЕНИЕ на команду СЛОЖЕНИЕ С ПЕРЕНОСОМ? Как изменяется при этом объем памяти, необходимый для размещения команды?

8. С одной стороны, команда СЛОЖЕНИЕ С ПЕРЕНОСОМ использует только значение разряда переноса регистра состояния, с другой – в ее выполнении участвуют разряды нулевого результата, переноса и отрицательного результата. Объясните, что имеется в виду в обоих случаях.

9. Почему в программе сложения чисел с повышенной точностью используются как команда СЛОЖЕНИЕ, так и команда СЛОЖЕНИЕ С ПЕРЕНОСОМ?

8.3. Команда ВЫЧИТАНИЕ

Ознакомившись с машинной арифметикой, становится ясно, что единственной незаменимой арифметической командой является команда СЛОЖЕНИЕ. Так, микропроцессор может выполнять вычитание путем формирования обратного кода вычитаемого, добавления 1 младшего разряда и сложения полученного таким образом числа с уменьшаемым. Однако быстрее и легче не создавать специальную программу вычитания, а возложить реализацию этой операции непосредственно на микропроцессор. Многие микропроцессоры располагают командами, которые не являются необходимыми, но существенно облегчают работу программиста. Команда ВЫЧИТАНИЕ как раз и относится к такой группе команд.

Ниже будут рассмотрены восемь команд ВЫЧИТАНИЕ, имеющихся в наборе команд нашего гипотетического микропроцес-

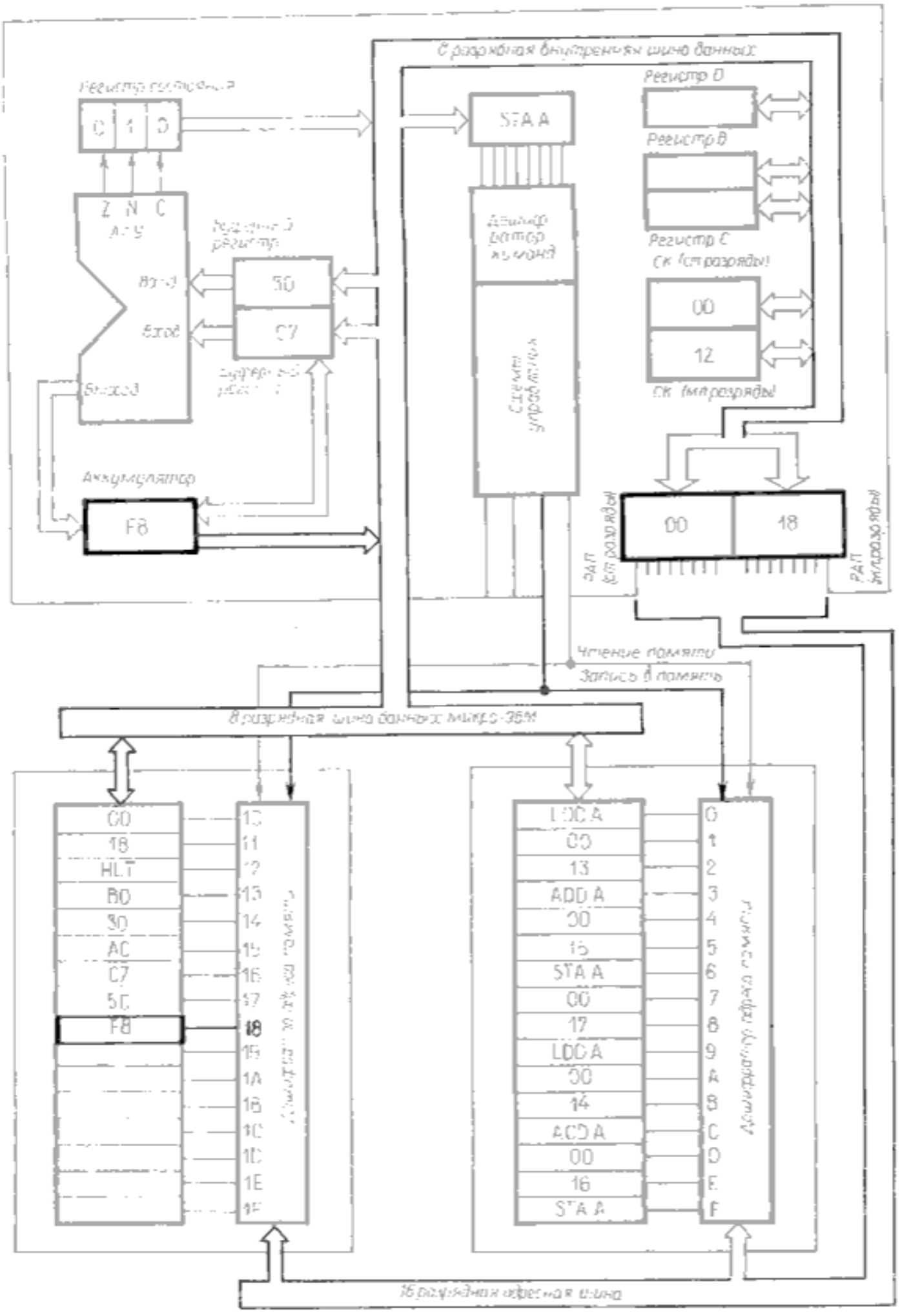


Рис 8 16 Запись старшего байта суммы в область памяти с адресом 0018

сора. Они похожи на команды СЛОЖЕНИЕ. Так, результат выполнения команды ВЫЧИТАНИЕ (разность), как и результат команды СЛОЖЕНИЕ, помещается в аккумулятор. Исходное значение уменьшаемого теряется. Как и при СЛОЖЕНИИ, при ВЫЧИТАНИИ соответствующим образом устанавливаются три разряда регистра состояния.

Все команды ВЫЧИТАНИЕ выполняются в дополнительном коде. Вычитание реализуется путем инвертирования значений всех разрядов вычитаемого, добавления к ним 1 младшего разряда и сложения полученного таким образом числа с уменьшаемым. Логическая 1 в старшем разряде является признаком отрицательного числа, а логический 0 – признаком положительного числа. Как мы выяснили, 8-разрядный дополнительный код дает возможность представлять числа в диапазоне от + 127 до – 128, включая 0.

Все рассматриваемые ниже команды вычитают данные, на которые имеется указание в команде, из данных, находящихся в аккумуляторе. Другими словами, в аккумуляторе всегда размещается уменьшаемое. Адрес, имеющийся в составе команды ВЫЧИТАНИЕ, задает местоположение вычитаемого.

Для выполнения вычитания служат следующие команды:

ВЫЧИТАНИЕ С РЕГИСТРОМ

SUB r $A - r \rightarrow A$

SUB r

При реализации команды ВЫЧИТАНИЕ С РЕГИСТРОМ содержимое регистра вычитается из содержимого аккумулятора (регистра А). Результат (разность) помещается в аккумулятор; исходное содержимое аккумулятора при этом теряется. Рассматриваемая команда является 1-байтовой командой, на выполнение которой затрачиваются два микроцикла процессора. Если результат выполнения команды является отрицательным или нулевым либо если возникает сигнал заема в старший разряд результата, соответствующие разряды регистра состояния устанавливаются в 1.

ВЫЧИТАНИЕ С ПАМЯТЬЮ ПРЯМОЕ

SUB M,Адрес $A - M \rightarrow A$

SUB M	1-й *байт*
СБ *адреса*	2-й *байт*
МБ *адреса*	3-й *байт*

По команде ВЫЧИТАНИЕ С ПАМЯТЬЮ ПРЯМОЕ содержимое области памяти вычитается из содержимого аккумулятора (регистра А). Второй и третий байты команды указывают на область памяти, в которой находится вычитаемое. Результат (разность) помещается в аккумулятор. Исходное содержимое аккумулятора теряется. Это 3-байтовая команда, для выполнения которой затрачиваются четыре микроцикла процессора. Если результат выполнения команды является отрицательным или нулевым либо если возникает заем в старший разряд результата, то соответствующие разряды регистра результата устанавливаются в 1.

ВЫЧИТАНИЕ С ПАМЯТЬЮ КОСВЕННОЕ

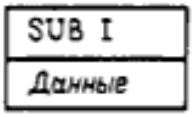
SUI M $A - M \rightarrow A$

SUI M

При реализации команды ВЫЧИТАНИЕ С ПАМЯТЬЮ КОСВЕННОЕ содержимое области памяти вычитается из содержимого аккумулятора (регистра А). Регистровая пара ВС указывает на область памяти, в которой находится вычитаемое. (Для того чтобы данная команда могла быть использована, в регистровой паре ВС должен содержаться необходимый адрес.) Результат (разность) помещается в аккумулятор. Исходное содержимое аккумулятора теряется. Это 1-байтовая команда, на выполнение которой затрачиваются три микроцикла процессора. Если результат выполнения команды является отрицательным или нулевым либо если возникает заем в старший разряд результата, то соответствующие разряды регистра состояния устанавливаются в 1.

ВЫЧИТАНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ

SUB I,Данные A – Данные → A



По команде ВЫЧИТАНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ содержимое второго байта команды вычитается из содержимого аккумулятора. Результат (разность) помещается в аккумулятор, при этом исходное содержимое аккумулятора теряется. Данная команда является 2-байтовой командой, выполнение которой занимает три рабочих цикла микропроцессора. Если результат выполнения команды является отрицательным или нулевым либо если возникает заем в старший разряд результата, соответствующие разряды регистра состояния устанавливаются в 1.

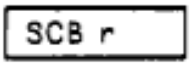
Аналогично тому, что имеются четыре команды СЛОЖЕНИЕ БЕЗ ПЕРЕНОСА и четыре команды СЛОЖЕНИЕ С ПЕРЕНОСОМ, существуют четыре команды ВЫЧИТАНИЕ, реализация которых не предусматривает заема, и четыре команды ВЫЧИТАНИЕ С ЗАЕМОМ. Заем происходит, когда вычитаемое оказывается больше уменьшаемого. При этом устанавливается в 1 разряд переноса (заема). Хотя при выполнении этих команд фактически имеет место заем, их называют ВЫЧИТАНИЕ С ПЕРЕНОСОМ, потому что в соответствии с результатом их выполнения устанавливается соответствующим образом разряд переноса в регистре состояния.

С помощью команды ВЫЧИТАНИЕ С ПЕРЕНОСОМ микропроцессор может выполнять вычитание с повышенной точностью. Команды ВЫЧИТАНИЕ С ЗАЕМОМ аналогичны командам ВЫЧИТАНИЕ, в которых, как известно, заем не производится, и отличаются лишь тем, что при их выполнении участвует текущий бит разряда переноса регистра состояния. Как и при реализации других команд ВЫЧИТАНИЕ, результаты загружаются в аккумулятор. Разряды отрицательного результата, нулевого результата и переноса устанавливаются в 1, если результат соответственно имеет знак минус, равен нулю или имеет место заем в старший разряд результата.

Ниже приводятся команды вычитания с переносом.

ВЫЧИТАНИЕ С РЕГИСТРОМ И ПЕРЕНОСОМ

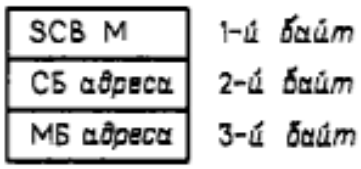
SCB r A – r – C → A



1 байт, два цикла

ВЫЧИТАНИЕ С ПАМЯТЬЮ И ПЕРЕНОСОМ ПРЯМОЕ

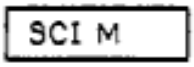
SCB M,Адрес A – M – C → A



3 байт, четыре цикла

ВЫЧИТАНИЕ С ПАМЯТЬЮ И ПЕРЕНОСОМ КОСВЕННОЕ

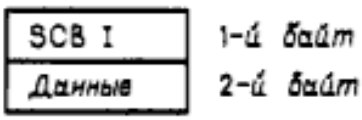
SCI M,Адрес A – M – C → A



1 байт, два цикла

ВЫЧИТАНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ И ПЕРЕНОСОМ

SCB I,Данные A – Данные – C → A



2 байт, три цикла

Вычитание, выполняемое над числами двойного формата, иллюстрирует использование и команды ВЫЧИТАНИЕ, и команды ВЫЧИТАНИЕ С ПЕРЕНОСОМ. Соответствующий пример приведен на рис. 8.17: десятичное число 12460 вычитается из десятичного числа 30812. Вычитание показано применительно к десятичной, шестнадцатеричной и двоичной формам представления чисел. Рисунок иллюстрирует выполнение вычитания отдельно над старшими и младшими байтами. Как можно видеть, после вычитания младших байтов для завершения вычитания необходим заем из старшего

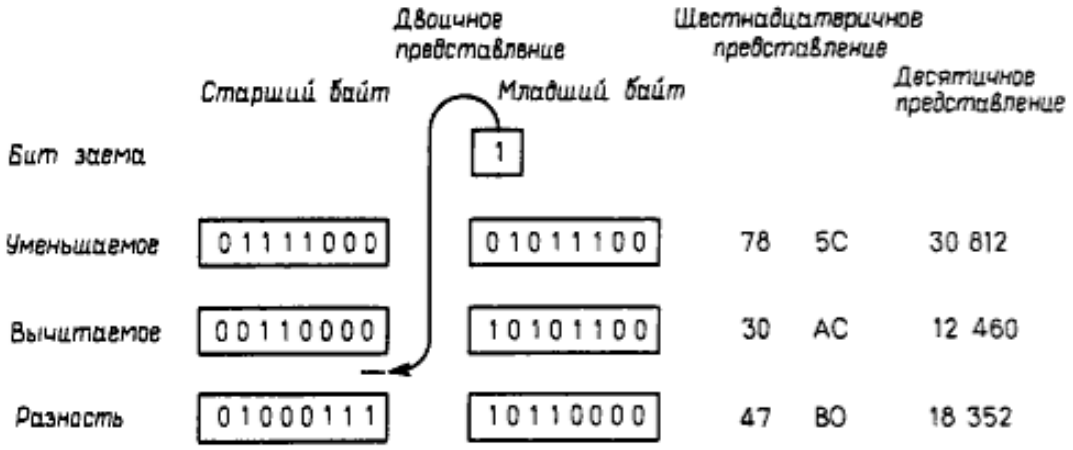


Рис. 8.17. Вычитание чисел двойного формата, в процессе которого при обработке младших байтов образуется сигнал заема. (Бит заема вычитается при обработке старших байтов.)

байта. Это означает, что в данном примере младший байт уменьшаемого меньше младшего байта вычитаемого. Следовательно, разряд переноса регистра состояния устанавливается в 1.

При выполнении обработки старших байтов используется команда ВЫЧИТАНИЕ С ПЕРЕНОСОМ. В этой операции участвуют уменьшаемое, вычитаемое и бит заема. Вычитаемое и бит разряда переноса регистра состояния отнимаются от уменьшаемого.

Блок-схема программы представлена на рис. 8.18, *а*, а схема распределения памяти – на рис. 8.18, *б*. В областях памяти 0013 и 0014 размещаются младший и старший байты уменьшаемого. В областях 0015 и 0016 находятся младший и старший байты вычитаемого. Области 0017 и 0018 отведены для младшего и старшего байтов разности.

На рис. 8.19 приведен листинг программы вычитания, которая напоминает программу сложения чисел двойного формата. Эта программа изменена таким образом, чтобы она выполняла вычитание чисел двойного формата. Проанализируем полученную при этом программу.

Первой в программе является команда LDD A, предназначенная для загрузки в аккумулятор значения области памяти 0013, в которой хранится младший байт уменьшаемого.

При выполнении второй команды значение области памяти 0015 вычитается из содержимого аккумулятора. В области с адресом 0015 хранится младший байт вычитаемого. Так как при вычитании имеет место

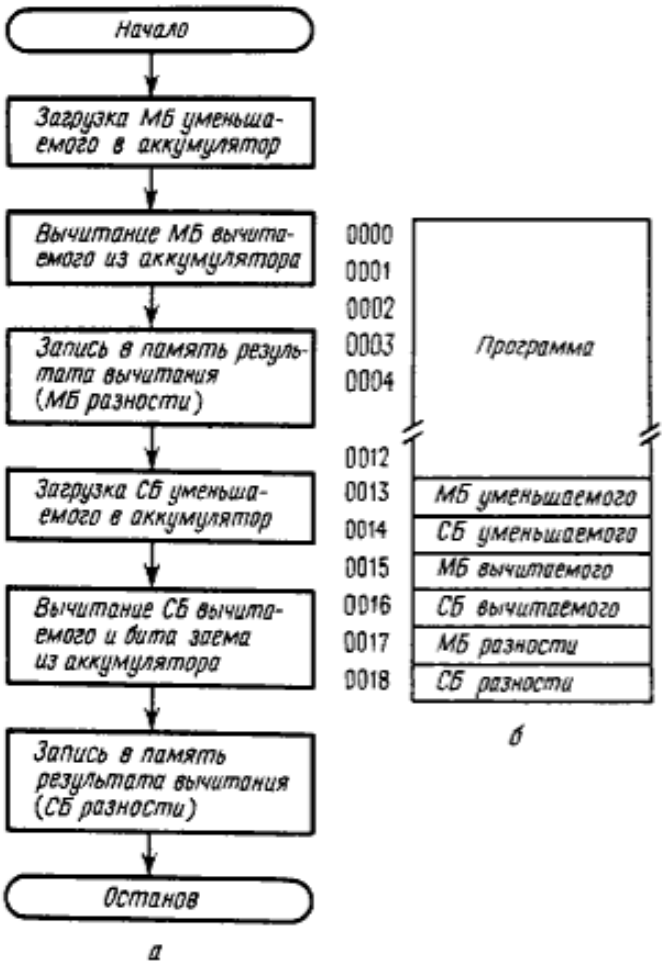


Рис. 8.18. *а* – Блок-схема и *б* – схема распределения памяти для программы вычитания чисел двойного формата. (Сравните эту блок-схему с блок-схемой программы сложения чисел двойного формата, представленной на рис. 8.9.)

заем, разряд переноса в регистре состояния устанавливается в 1. Для данной операции вычитания использована команда SUB M. При ее выполнении нет необходимости использовать значение заема (переноса), имев-

Адрес области памяти	Содержимое области памяти	Комментарий
0000	LDD A	ЗАГРУЗКА АККУМУЛЯТОРА ПРЯМАЯ
0001	00	Адрес младшего байта
0002	13	уменьшаемого
0003	SUB M	ВЫЧИТАНИЕ С ПАМЯТЬЮ ПРЯМОЕ
0004	00	Адрес младшего байта
0005	15	вычитаемого
0006	STA A	ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ ПРЯМАЯ
0007	00	Адрес младшего байта разности
0008	17	
0009	LDD A	ЗАГРУЗКА АККУМУЛЯТОРА ПРЯМАЯ
000A	00	Адрес старшего байта
000B	14	уменьшаемого
000C	SCB M	ВЫЧИТАНИЕ С ЗАЕМОМ
000D	00	Адрес старшего байта
000E	16	вычитаемого
000F	STA A	ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ ПРЯМАЯ
0010	00	Адрес старшего байта
0011	18	разности
0012	HLT	ОСТАНОВ
0013	5C	Младший байт — Уменьшаемое
0014	78	Старший байт — (30 812)
0015	AC	Младший байт — Вычитаемое
0016	30	Старший байт — (12 460)
0017	B0	Младший байт — Разность
0018	47	Старший байт — (18 352)

Рис. 8.19. Листинг программы вычитания чисел двойного формата.

шего место в результате предшествующих операций. Так как это первая операция вычитания, входящая в последовательность действий по обработке данных двойного формата, значение разряда переноса в регистре состояния, установленное предыдущей операцией, во внимание не принимается. Если бы в данном случае была использована команда SCB M, пришлось бы вводить дополнительные команды для установки в 0 разряда переноса. Прибегнув к команде вычитания, в выполнении которой бит переноса не участвует, можно обходиться без таких лишних команд.

Результат вычитания младших байтов находится теперь в аккумуляторе. При выполнении третьей команды содержимое аккумулятора записывается в область памяти (0017), отведенную для хранения разности. Для этой цели применена команда STA A.

В качестве четвертой команды программы вторично используется команда LDD A. С ее помощью в аккумулятор загружается старший байт уменьшаемого.

В соответствии с пятой командой старший байт вычитаемого вычитается из старшего байта уменьшаемого. Старший байт вычитаемого хранится в области 0016. Для вычитания значения области 0016 из содержимого аккумулятора применена команда SCB M. При выполнении этой команды из результата вычитается также и значение

разряда переноса. В данном случае разряд переноса в регистре состояния установлен в 1, так как в ходе предшествующего вычитания младших байтов был выработан заем из старшего байта. Старший байт разности, являющийся результатом выполнения команды SCB M, помещается в аккумулятор.

В качестве шестой команды программы вновь используется команда STA A, служащая для записи старшего байта разности в область памяти 0018. Теперь разность чисел двойного формата полностью располагается в областях 0017 и 0018.

Рассмотренный пример выполнения арифметической операции над числами двойного формата показывает, зачем в набор команд микропроцессора включаются однотипные и в то же время разные команды. Различие, существующее между командами ВЫЧИТАНИЕ С ПЕРЕНОСОМ и ВЫЧИТАНИЕ, позволяет обойтись без лишней команды, которая потребовалась бы для сброса разряда переноса в регистре состояния. Если бы в начале программы была использована команда CLA A (ОЧИСТКА АККУМУЛЯТОРА), для выполнения программы потребовались бы два дополнительных микроцикла процессора. В данном примере удалось обойтись без этой лишней команды, поскольку микропроцессор располагает гибким набором команд, содержащим команды вычитания двух типов. В такой ситуации более эффективным оказывается применение команды ВЫЧИТАНИЕ, не учитывающей значение разряда переноса.

Задания для самопроверки

10. Измените блок-схему программы, представленную на рис. 8.18, таким образом, чтобы она выполняла вычитание чисел тройного формата.

11. Какие признаки результата операции ВЫЧИТАНИЕ вызывают установку в 1 разряда переноса регистра состояния?

12. Какие признаки результата операции ВЫЧИТАНИЕ вызывают установку в 1 разряда нулевого результата в регистре состояния?

13. Какие признаки результата операции ВЫЧИТАНИЕ вызывают установку в 1 разряда отрицательного результата в регистре состояния?

14. Что необходимо сделать для представления числа в двоичном формате без знака, если при выполнении операции ВЫЧИТАНИЕ установлен в 1 разряд отрицательного результата в регистре состояния?

8.4. Команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ

Наличие команды ДЕСЯТИЧНАЯ КОРРЕКЦИЯ позволяет микропроцессору выполнять некоторые виды обработки двоично-десятичных чисел. Конечно, эта возможность может быть предоставлена микропроцессору и программным путем. Однако, как мы уже могли убедиться, наличие специальных команд может помочь сократить общее количество шагов выполнения программы.

Почему же необходимо, чтобы микропроцессор обрабатывал двоично-десятичные числа? Ответ на этот вопрос связан с областями применения микропроцессоров. Часто последние используются в простых системах невычислительного характера, для которых характерны ввод и вывод информации в двоично-десятичном коде (BCD).

Например, микропроцессоры применяются сейчас в электронных счетных системах. Входной информацией для микропроцессора являются при этом двоично-десятичные данные, поступающие от быстродействующих декадных счетчиков. Эти счетчики служат для подсчета входных сигналов прибора. Микропроцессор выполняет обработку двоично-десятичных данных, такую, как преобразование временных интервалов в частоту, определение среднего значения, автоматическое масштабирование и т.д. Выходные данные микропроцессора, предназначенные для управления системой индикации электронного счетчика, также являются двоично-десятичными. Структурная схема электронной пересчетной системы приведена на рис. 8.20. Как ввод, так и вывод данных из однокристальной микро-ЭВМ осуществляется в двоично-десятичном коде.

Из этого примера становятся понятны причины, которые делают рациональным

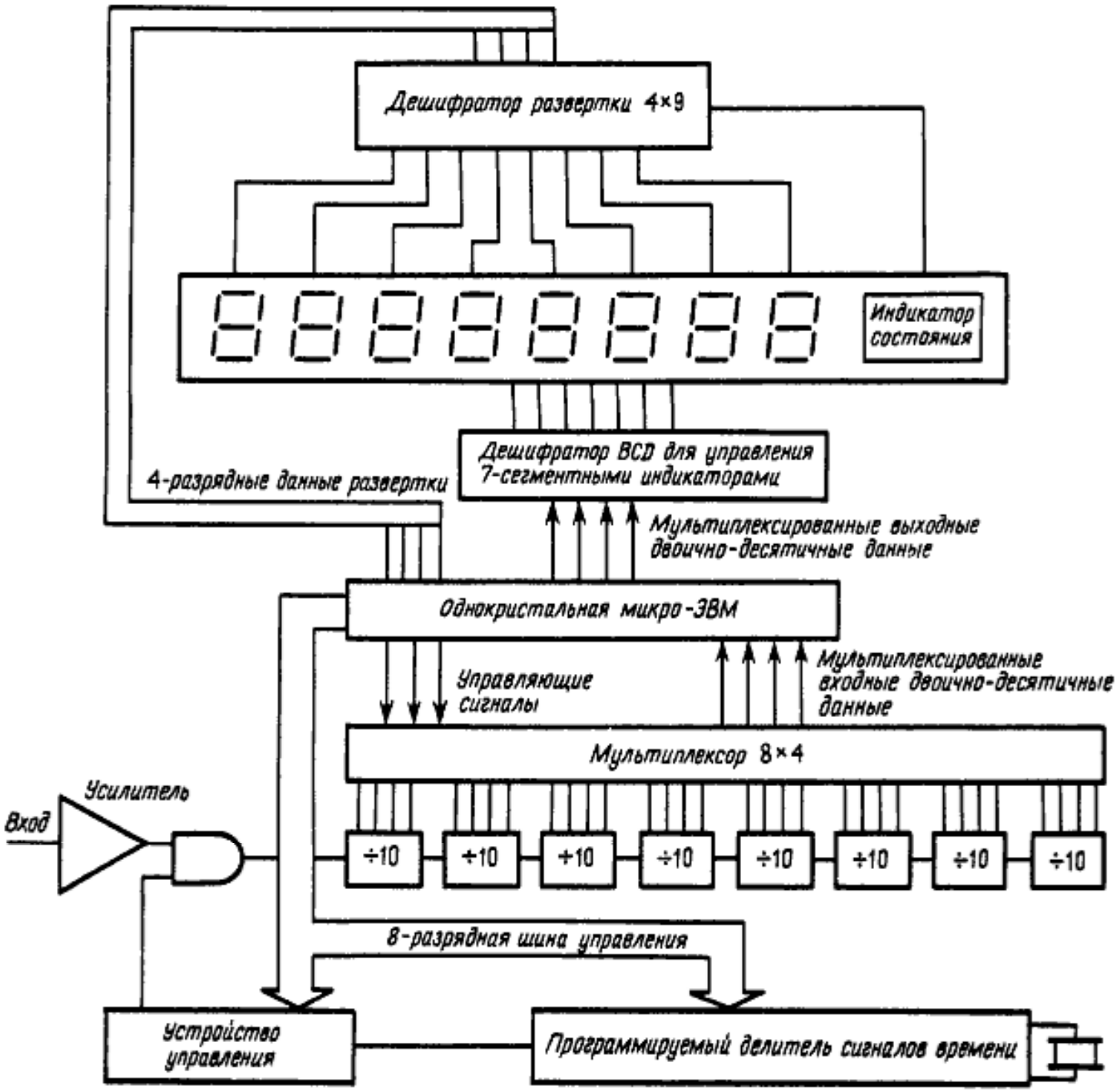


Рис. 8.20. Часы (измеритель временны́х интервалов) и частотомер на основе микро-ЭВМ. (Микро-ЭВМ управляет прибором, определяя положение десятичных точек, показания индикаторов и опорные частоты. Она выполняет также вычисления над данными, такие как определение обратных и средних значений.)

возложение на микропроцессор функций обработки двоично-десятичных данных. Коль скоро микропроцессор должен выполнять операции над двоично-десятичными данными, желательно освободить его от дополнительной работы, связанной с преобразованием двоично-десятичных данных в двоичную форму. Имеется много других приложений, в которых целесообразно иметь на входе и выходе микропроцессора двоично-десятичные данные, вместо того чтобы выполнять их преобразование в двоичные данные и обратное преобразование. Так, например, терминалы, которыми снабжены торговые точки (универмаги, кафетерии и т. д.), обычно снабжены клавиатурой в двоично-десятичном коде и соответствующим устройством отображения данных. В высокочастотных печах и автономных электронных играх – предметах бытового пользования – также применяются двоично-десятичные данные.

Команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ позволяет упростить обработку двоично-де-

сятичных данных в микропроцессорной системе. Прежде чем приступить к описанию этой команды, выясним, какие действия она выполняет.

При использовании двоичных чисел для кодирования десятичных данных необходимо иметь возможность представлять цифры от 0 до 9. Ясно, что для изображения цифры 9_{10} в виде 1001_2 требуется 4 бит. Таким образом, для представления любой десятичной цифры необходимы 4 бит. Однако все 4-разрядные двоичные числа, превышающие 1001_2, недопустимы в двоично-десятичной системе счисления. Значит, если сумма двоично-десятичных цифр не является числом, меньшим или равным 9_{10}, она представляет собой недопустимое для использования двоично-десятичное число. Команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ служит для исправления результатов двоично-десятичной обработки, представляющих собой числа, бóльшие чем 1001_2.

Команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ имеет следующий формат:

ДЕСЯТИЧНАЯ КОРРЕКЦИЯ АККУМУЛЯТОРА

DAD A — A → Байт, содержащий два двоично-десятичных числа

DAD A

1 байт, два цикла

Эта команда выполняется сразу же за сложением двух двоично-десятичных чисел, после выполнения которого сумма находится в аккумуляторе. Результат сложения обрабатывается командой ДЕСЯТИЧНАЯ КОРРЕКЦИЯ, чтобы обеспечить его представление в виде допустимых двоично-десятичных цифр.

Команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ устанавливает разряды нулевого и отрицательного результатов регистра состояния в соответствии с признаками результата. Разряд переноса в регистре состояния устанавливается в 1, когда значение старшей десятичной цифры превышает 9 (1001_2).

Действие команды ДЕСЯТИЧНАЯ КОРРЕКЦИЯ иллюстрирует следующий пример. На рис. 8.21, *а* показано сложение двух 4-разрядных десятичных чисел. Как отмечено выше, двоично-десятичный формат является широко распространенным средством представления чисел в электронной аппаратуре. На рис. 8.21, *б* приведена данная операция сложения в двоично-десятичной форме. Результатом обычного сложения является в данном случае число, не являющееся ни двоично-десятичным, ни двоичным. Напомним еще раз, что любое 4-разрядное число, большее чем 1001_2, недопустимо в двоично-десятичной системе счисления.

Коррекция результата с целью приведе-

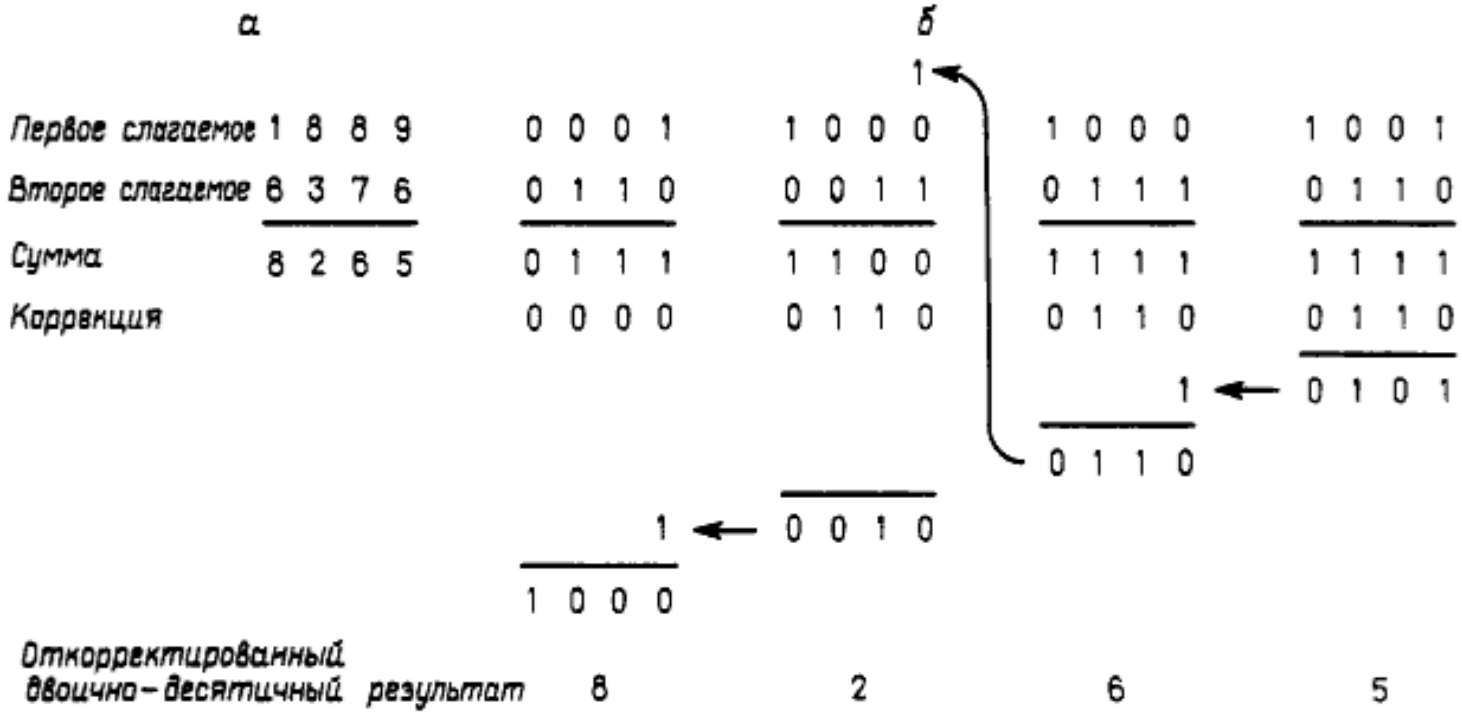


Рис. 8.21. *а* – Сложение двух 4-разрядных десятичных чисел; *б* – сложение двоично-десятичных данных.

ния его к двоично-десятичной форме также показана на рис. 8.21, *б*. К каждой двоично-десятичной цифре результата, превышающей 1001_2, прибавляется число 0110_2. Обработка начинается с пары двоично-десятичных цифр, представляющих младшие разряды результата. Перенос, возникающий при коррекции любого двоично-десятичного разряда, должен быть учтен при коррекции соседней двоично-десятичной цифры. Как можно видеть, таким путем двоично-десятичная сумма приводится к пригодному виду. В каждом ее разряде содержится теперь допустимая двоично-десятичная цифра.

Блок-схема программы, реализующей сложение 4-разрядных двоично-десятичных чисел, приведена на рис. 8.22. Эта программа во многом схожа с рассмотренной выше программой сложения чисел двойного формата и отличается от нее главным образом тем, что после каждой операции сложения выполняется команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ.

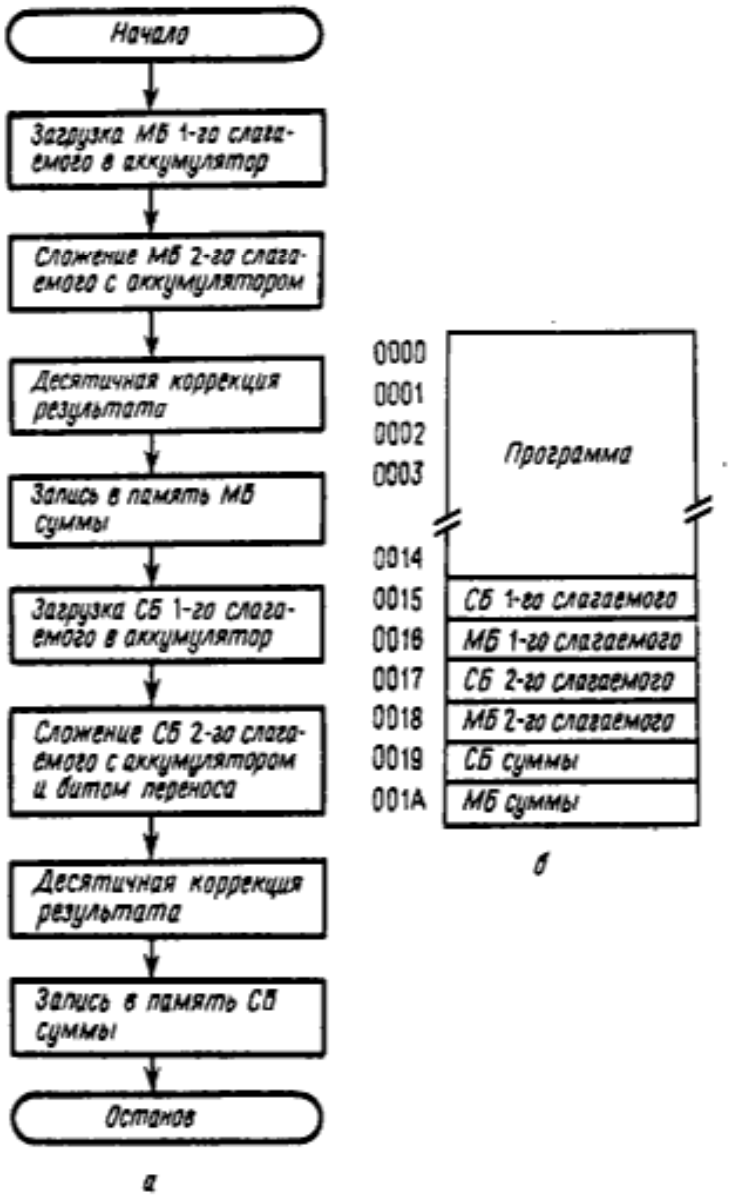


Рис. 8.22. Блок-схема программы и схема распределения памяти для сложения 4-разрядных двоично-десятичных чисел.

Листинг этой программы представлен на рис. 8.23. Эта программа только на несколько команд длиннее программы сложения двоичных чисел двойного формата. Программа позволяет обрабатывать двоично-десятичные числа, результат сложения которых лежит в диапазоне от 0 до 9999. Проведя анализ двух программ, можно сделать вывод о преимуществах использования обычного двоичного сложения в тех случаях, когда его применение допустимо. При обычном сложении чисел двойного формата результат может находиться в пределах от 0 до 65 535, причем количество команд в программе, реализующей такое сложение, меньше, чем в программе сложения двоично-десятичных чисел. За возможность выполнения обработки двоично-десятичной информации приходится чем-то поступиться.

По первой команде младший байт первого слагаемого загружается в аккумулятор. Обратите внимание, что этот младший байт содержит две двоично-десятичные цифры. Это две младшие цифры первого слагаемого – двоично-десятичные цифры 8 и 9. Если в одном байте содержатся две двоично-десятичные цифры, то формат представления двоично-десятичного числа называется *упакованным*.

Вторая команда программы служит для прибавления младшего байта второго слагаемого к младшему байту первого слагаемого. Младший байт второго слагаемого состоит из двоично-десятичных цифр 7 и 6. На рис. 8.21 видно, что после выполнения этой операции сложения содержимое аккумулятора представляет собой байт 1111 1111. Конечно, это не двоично-десятичные цифры.

Третьей в программе является команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ. Она заменяет содержимое аккумулятора 1111 1111 на 0110 0101. Кроме того, по этой команде устанавливается в 1 разряд переноса в регистре состояния, так как в результате выполнения десятичной коррекции возникает перенос в старший байт.

По четвертой команде откорректиро-

Адрес области памяти	Содержимое области памяти	Комментарий
0000	LDD A	ЗАГРУЗКА АККУМУЛЯТОРА ПРЯМАЯ
0001	00	Адрес младшего байта первого слагаемого
0002	16	
0003	ADD M	СЛОЖЕНИЕ С ПАМЯТЬЮ ПРЯМОЕ
0004	00	Адрес младшего байта второго слагаемого
0005	18	
0006	DAD	ДЕСЯТИЧНАЯ КОРРЕКЦИЯ
0007	STA A	ЗАПИСЬ В ПАМЯТЬ АККУМУЛЯТОРА ПРЯМАЯ
0008	00	Адрес младшего байта суммы
0009	1A	
000A	LDD A	ЗАГРУЗКА АККУМУЛЯТОРА ПРЯМАЯ
000B	00	Адрес старшего байта первого слагаемого
000C	15	
000D	ACD M	СЛОЖЕНИЕ С ПЕРЕНОСОМ
000E	00	Адрес старшего байта второго слагаемого
000F	17	
0010	DAD	ДЕСЯТИЧНАЯ КОРРЕКЦИЯ
0011	STA A	ЗАПИСЬ В ПАМЯТЬ АККУМУЛЯТОРА ПРЯМАЯ
0012	00	Адрес старшего байта суммы
0013	19	
0014	HLT	ОСТАНОВ
0015	00011000	Старший байт Первое слагаемое
0016	10001001	Младший байт (1889)
0017	01100011	Старший байт Второе слагаемое
0018	01110110	Младший байт (6376)
0019	10000010	Старший байт Сумма (8265)
001A	01100101	Младший байт

Рис. 8.23. Листинг программы сложения двух 16-разрядных упакованных двоично-десятичных чисел.

ванные двоично-десятичные цифры 6 и 5 записываются в область памяти 001A. Это две младшие цифры суммы.

При выполнении пятой команды две старшие цифры первого слагаемого загружаются из области памяти в аккумулятор.

С помощью шестой команды две старшие цифры второго слагаемого складываются с двумя старшими цифрами первого слагаемого. Из листинга программы следует, что для этой цели используется команда СЛОЖЕНИЕ С ПЕРЕНОСОМ. Применена именно эта команда, так как при сложении должно быть учтено значение переноса, выработанного в результате предыдущей команды ДЕСЯТИЧНАЯ КОРРЕКЦИЯ. Этот перенос находится в соответствующем разряде регистра состояния.

Обратившись вновь к рис. 8.21, можно увидеть, что после выполнения команды сложения с переносом в аккумуляторе содержатся двоичные числа 0111 и 1100. Ни одно из них не является двоично-десятичной цифрой. Опять применяется команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ. Результатом ее выполнения являются числа 1000 и 0010. Это двоично-десятичные цифры 8 и 2.

Таким образом, в рассматриваемой программе широко используется команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ. Если оба числа, являющиеся результатом сложения, представляют собой допустимые двоично-десятичные цифры, то коррекция не производится. Значит, в начале выполнения команды ДЕСЯТИЧНАЯ КОРРЕКЦИЯ выясняется, есть ли в составе результата двоичные числа, превышающие 1001_2.

Для записи полученных двоично-десятичных цифр в область памяти 0019 еще раз используется команда ЗАПИСЬ В ПАМЯТЬ. Таким образом, в память записывается старший байт суммы. Теперь в областях 0019 и 001A находится двоично-десятичная сумма 8265.

Назначением команды ДЕСЯТИЧНАЯ КОРРЕКЦИЯ является введение поправки в результат сложения двух двоично-десятичных чисел, полученный с помощью двоичного сумматора. Эта команда не может быть использована для коррекции результата вычитания, выполненного над двоично-десятичными числами как над простыми двоичными числами. Она применяется для внесения поправок только в результаты выполнения двоичного сложения.

Существуют программные средства, позволяющие выполнять над двоично-десятичными числами любые арифметические операции. Имеются, кроме того, и программы для преобразования двоично-десятичных чисел в двоичные числа и для обратного преобразования. Очевидно, что использование этих средств приводит к необходимости создания программ, значительно превышающих по объему одну команду.

Задания для самопроверки

15. Почему при наличии команды ДЕСЯТИЧНАЯ КОРРЕКЦИЯ не возникает необходимости в команде ДЕСЯТИЧНАЯ КОРРЕКЦИЯ С ПЕРЕНОСОМ?

16. Команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ применима только при сложении двух двоично-десятичных чисел. Можно ли ее использовать при вычитании двоично-десятичных чисел?

17. Команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ должна анализировать результат предыдущей операции, находящийся в аккумуляторе, и вводить поправки в соответствии с четырьмя возможными комбинациями чисел, составляющих этот результат. Две из этих комбинаций следующие: 1) младшая двоично-десятичная цифра превышает 1001_2, а старшая меньше или равна 1001_2; в этом случае для коррекции результата к содержимому аккумулятора прибавляется число 0110_2; 2) обе двоично-десятичные цифры превышают 1001_2; для коррекции к результату прибавляется число $0110\ 0110_2$. Приведите две остальные возможные комбинации.

18. Изобразите блок-схему программы и схему распределения памяти, на которой показаны области, отводимые под слагаемые и сумму, для ситуации, когда результаты сложения двух двоично-десятичных чисел лежат в диапазоне от 0 до 999 999.

19. Какова разрядность двоичных чисел, необходимая для реализации задачи, сформулированной в вопросе 18? Одинаково ли количество байтов, необходимое для представления слагаемых в виде двоичных чисел без знака и в виде двоично-десятичных чисел, если предполагается использование 8-разрядного микропроцессора?

20. Если слово данных микропроцессора содержит более одной двоично-десятичной цифры, то мы говорим, что двоично-десятичное число представлено в упакованном формате. Так, например, в 8-разрядном микропроцессоре в одном слове упакованы две двоично-десятичные цифры. Как соотносятся диапазоны представления данных в виде двоичных слов двойного формата и в виде упакованных двоично-десятичных чисел двойного формата в 16-разрядном микропроцессоре?

8.5. Команды ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ и ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ

Команды ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ и ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ являются в сущности специальными арифметическими командами. При выполнении команды ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ прибавляется 1 к текущему содержимому регистра. По команде

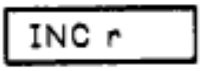
ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ из текущего содержимого регистра вычитается 1. Эти команды используются в ситуациях, когда необходимо подсчитывать число каких-то событий, т. е. когда регистр следит за тем, сколько раз была выполнена определенная операция.

Хотя рассматриваемый здесь гипотетический микропроцессор располагает только командами положительного и отрицательного приращения содержимого регистров, многие микропроцессоры предоставляют в распоряжение пользователя аналогичные команды для обработки содержимого областей памяти.

Команды ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ и ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ – это 1-байтовые команды, на выполнение каждой из которых затрачиваются два микроцикла процессора. Результаты выполнения этих команд не сказываются на значении разряда переноса регистра состояния.

ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ РЕГИСТРА

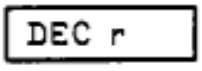
INC r $r + 1 \rightarrow r$



По команде ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ РЕГИСТРА двоичная 1 прибавляется к содержимому указанного в команде регистра. Если результат ее выполнения содержит 1 в старшем разряде или равен нулю, устанавливаются в 1 соответствующие разряды регистра состояния.

ОТРИЦАТЕЛЬНОЕ ПРИРАЩЕНИЕ РЕГИСТРА

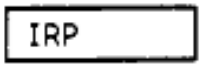
DEC r $r - 1 \rightarrow r$



При выполнении команды ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ РЕГИСТРА двоичная 1 вычитается из содержимого указанного в команде регистра. Во всех других отношениях эта команда аналогична команде ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ РЕГИСТРА. Если результат ее выполнения содержит 1 в старшем разряде или равен нулю, устанавливаются в 1 соответствующие разряды регистра состояния.

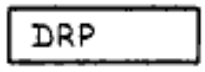
ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ РЕГИСТРОВОЙ ПАРЫ

IRP $BC + 1 \rightarrow BC$



ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ РЕГИСТРОВОЙ ПАРЫ

DRP $BC - 1 \rightarrow BC$



Команды IRP и DRP предназначены для обработки содержимого регистровой пары BC. Единственное их отличие от рассмотренных выше команд положительного и отрицательного приращения содержимого регистра состоит в том, что они оперируют содержимым пары регистров BC, рассматриваемой как единый 16-разрядный регистр.

Это означает, что положительное приращение содержимого регистров BC от 00FF до 0100 или отрицательное приращение от 0100 до 00FF может быть выполнено автоматически. При этом нет необходимости предусматривать специальную команду, которая выясняла бы наличие переноса между регистрами B и C. При использовании команд положительного и отрицательного приращения содержимого регистров перенос из регистра B в регистр C отсутствует. В зависимости от применяемой команды эти регистры могут рассматриваться либо как два 8-разрядных, либо как один 16-разрядный. При положительном или отрицательном приращении содержимого регистровой пары результат операции отражается только на значении разряда нулевого результата регистра состояния. Этот разряд устанавливается в 1, когда по окончании выполнения команды положительного или отрицательного приращения содержимого регистровой пары все ее 16 разрядов оказываются равными 0.

Команда ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ часто используется для установления того факта, что определенная часть программы была выполнена заданное количество раз. Эта задача может быть решена различными путями. Можно воспользоваться командой ПРИРАЩЕНИЕ ПОЛО-

ЖИТЕЛЬНОЕ. Это означает, что каждый раз по достижении определенной точки программы с помощью данной команды увеличивается на 1 содержимое некоторого регистра. Затем оно проверяется с помощью команды СРАВНЕНИЕ. Если в результате выполнения команды СРАВНЕНИЕ устанавливается в 1 разряд нулевого результата в регистре состояния, по этому признаку можно перейти к любой заданной части программы.

Команда ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ позволяет решить ту же задачу, затратив меньшее количество команд. Вначале в регистр заносится некоторое двоичное число, которое равно желаемому количеству выполнений фрагмента программы. Затем при каждом проходе программы через определенную точку содержимое регистра уменьшается на 1. После выполнения команды отрицательного приращения команда ПЕРЕХОД, ЕСЛИ НУЛЬ проверяет, не привело ли уменьшение содержимого регистра на 1 к установке в 1 разряда нулевого результата в регистре состояния. Если это так, то происходит переход к заданной области. Если же выясняется, что уменьшение на 1 содержимого регистра не вызвало установки в 1 разряда нулевого результата в регистре состояния, выполнение программы продолжается. Данный подход к построению программы позволяет уменьшить ее длину на одну команду.

Используя команду ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ для реализации счетчика программным путем, необходимо очень внимательно отнестись к вопросу о местоположении этой команды в программе. Так, подсчитываемое число операций различно в зависимости от того, до или после подлежащего счету действия выполняется команда ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ.

Предположим, что необходимо подсчитать, сколько раз используется команда ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ. На рис. 8.24, *а* и *б* представлены два способа организации этого подсчета. Программа, приведенная на рис. 8.24, *а*, начинается с команды загрузки в регистр В непосредственных данных. В регистр В загружается число 03, так как необходимо выполнить команду ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ три раза.

Как следует из приведенной программы, после загрузки регистра В выполняются другие команды, после которых реализуется команда ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ. Очередной является команда отрицательного приращения содержимого регистра В. Если после этой команды все разряды регистра В приобретают значение логического 0, то происходит переход к последней команде программы.

В программе, показанной на рис. 8.24, *а*, команда ЗАПИСЬ В ПАМЯТЬ используется три раза. После выполнения этой команды в третий раз все разряды регистра В приобретают значение логического 0. По этой причине реализуется команда ПЕРЕХОД, ЕСЛИ НУЛЬ, и программа завершается.

Программа, представленная на рис. 8.24, *б*, организована по-другому. Начинается она аналогичным образом, т.е. с загрузки в регистр В числа 03. После выполнения ряда команд производится отрицательное приращение содержимого регистра В. Затем реализуется команда ПЕРЕХОД, ЕСЛИ НУЛЬ, за которой следует первая команда ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ.

После вторичного использования команды ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ имеет место несколько других команд. Непосредственно перед выполнением в третий раз команды ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ осуществляется в третий раз отрицательное приращение содержимого регистра В; все разряды регистра В приобретают при этом значение логического 0. Очередной в программе является команда ПЕРЕХОД, ЕСЛИ НУЛЬ. После этой команды микропроцессор сразу же переходит к заключительной команде программы. Команда ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ в третий раз так и не выполняется. Происходит так потому, что в данном случае команды ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ и ПЕРЕХОД, ЕСЛИ НУЛЬ находятся перед командой ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ, а не после нее.

Рассмотрение данного примера позволяет установить два обстоятельства. Во-первых, он дает представление о том, каким образом используется команда ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ для реализации

Последовательность		Команда		Регистр B	Количество выполнений команды ЗАПИСЬ В ПАМЯТЬ
1		LDA	B,03	03	
.		.		.	
.	PGM	.		.	
.		.		.	
17		STI	A	03	1
18		DEC	B	02	
19		JZ	99	02	
.		.		.	
.	PGM	.		.	
.		.		.	
39		STI	A	02	2
40		DEC	B	01	
41		JZ	99	01	
.		.		.	
.		.		.	
.		.		.	
70		STI	A	02	3
71		DEC	B	00	
72		JZ	99		
99		HLT			

а

1		LDA	B,03	03	
.		.		.	
.	PGM	.		.	
.		.		.	
17		DEC	B	02	
18		JZ	99	02	
19		STI	A	02	1
.		.		.	
.	PGM	.		.	
.		.		.	
39		DEC	B	01	
40		JZ	99	01	
41		STI	A	01	2
.		.		.	
.	PGM	.		.	
.		.		.	
70		DEC	B	00	
71		JZ			
72		STI	A		
99		HLT			

б

Рис. 8.24. Два варианта программирования функционирования счетчика, в котором реализован принцип уменьшения содержимого: *а* – команда ЗАПИСЬ В ПАМЯТЬ выполняется до останова программы три раза; *б* – команда ЗАПИСЬ В ПАМЯТЬ выполняется до останова программы два раза.

счетчика программным путем. Этот счетчик следит за тем, сколько раз были выполнены определенные действия. Его показания дают возможность закончить выполнение программы после повторения этих действий заданное число раз. Во-вторых, пример показывает, насколько важно тщательно продумать логику построения программы. В зависимости от местоположения в программе команды ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ изменяется количество выполняемых команд.

В зависимости от того, где находится команда в программе, имеются два выхода. Можно использовать программу, приведенную на рис. 8.24, *а*. Кроме того, существует возможность изменить данные, входящие в состав первой команды программы, представленной на рис. 8.24, *б*. Достаточно загрузить в регистр B начальное значение 04, тогда как практически соответствующий фрагмент программы должен быть выполнен лишь три раза. Тем самым компенсируется потеря одного из нужных циклов его выполнения.

В некоторых рассмотренных выше примерах была использована команда ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ РЕГИСТРОВОЙ ПАРЫ. Как отмечено выше, команда ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ РЕГИСТРОВОЙ ПАРЫ отличается от нее лишь тем, что осуществляет не прибавление, а вычитание двоичной 1 из содержимого регистра. Разумеется, использование 16-разрядного регистра BC для адресации памяти является одним из наиболее типичных его применений.

Как правило, к соседним областям памяти необходимо последовательно обращаться в порядке возрастания их адресов. Однако иногда приходится организовывать доступ к соседним областям в порядке убывания адресов. Для этой цели служит команда ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ РЕГИСТРОВОЙ ПАРЫ. Эта команда также может быть применена для реализации простого счетчика.

Предположим, например, что нам нужно записывать данные в буферную память, имеющую объем 24 576 байт. В любой момент времени необходимо знать, сколько свободных областей размером в байт еще остается в памяти. Простейший способ решения этой задачи заключается в использовании регистровой пары BC в качестве счетчика. Соответствующая программа организуется аналогично программе, представлен-

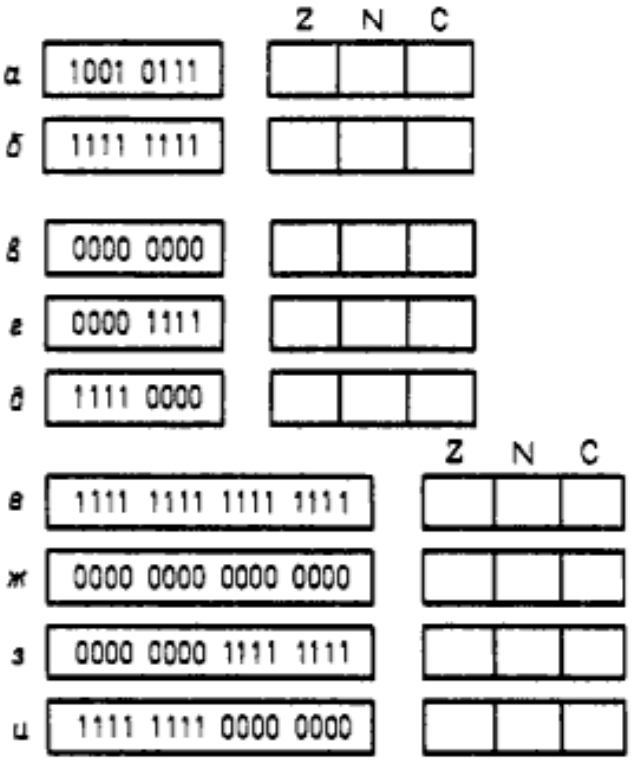
ной на рис. 8.24, и отличается от нее лишь значением числа, загружаемого в регистровую пару BC в начале программы. При записи в буферную память очередного байта программа осуществляет отрицательное приращение числа, находящегося в регистровой паре BC. Если впоследствии данные будут изъяты из буфера, например для записи на магнитную ленту, программа обеспечит положительное приращение содержимого регистров BC на величину, равную количеству изъятых из буфера байтов. При необходимости узнать, сколько свободных областей остается в буфере, достаточно обратиться к регистровой паре BC.

шаться на определенное значение, используются в качестве машинных часов. С этой целью создается небольшая подпрограмма, с помощью которой можно зафиксировать, когда заканчивается определенный интервал времени. Как организуются такие машинные часы? Какой характеристикой микропроцессора определяется их точность?

24. Укажите способ использования регистра, над содержимым которого может выполняться отрицательное приращение, для выполнения операции умножения. Приведите блок-схему простой программы умножения двух 8-разрядных чисел с использованием такого регистра. Укажите недостатки данного метода умножения.

Задания для самопроверки

21. Покажите, как изменится содержимое рабочего регистра и регистра состояния после выполнения команды ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ РЕГИСТРА (РЕГИСТРОВОЙ ПАРЫ):



22. Покажите, как изменится содержимое рабочего регистра и регистра состояния (см. рисунок к п. 21) после выполнения команды ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ РЕГИСТРА (РЕГИСТРОВОЙ ПАРЫ).

23. Иногда счетчики, содержимое которых может увеличиваться или умень-

Упражнения

8.1. К какому типу команд микропроцессора [а) логической, б) арифметической, в) для переноса, г) для пересылки] относится команда СЛОЖЕНИЕ?

8.2. В команде ADD r мнемоническая запись ADD служит для обозначения кода операции. Укажите, обозначает буква r:

а) результат арифметической операции, б) содержимое аккумулятора, в) адрес или г) бит переноса?

8.3. Является команда СЛОЖЕНИЕ С ПАМЯТЬЮ ПРЯМОЕ рассматриваемого здесь микропроцессора а) 1-, б) 2-, в) 3- или г) 4-байтовой командой?

8.4. При выполнении команды СЛОЖЕНИЕ указываемые в ней данные прибавляются к содержимому аккумулятора. Что из нижеперечисленного характерно для исходного содержимого аккумулятора: исходное содержимое а) представляет собой слагаемое, б) представляет собой сумму, в) записывается в память, г) теряется?

8.5. Какой из разрядов в регистре состояния [а) нулевого результата, б) переноса, в) отрицательного результата] не устанавливается в 1 при выполнении команды СЛОЖЕНИЕ?

8.6. Уточните, что происходит с результатом выполнения команды СЛОЖЕНИЕ С ПЕРЕНОСОМ:

а) формируется с участием разряда переноса регистра состояния;

б) определяет значение разряда переноса по окончании операции;

в) формируется с участием разряда заема регистра состояния, или перечисленные здесь действия не имеют к нему отношения.

8.7. Укажите, числа со знаком какого формата [а) одинарного, б) двойного, в) тройного, г) уче-

тверенного] необходимо применить для представления данных при работе с 8-разрядным микропроцессором, десятичные результаты выполнения операций которого лежат в диапазоне от $-1\,000\,000$ до $+1\,000\,000$.

8.8. Команда СЛОЖЕНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ имеет длину 2 байт. Какой величине равен этот параметр для команды СЛОЖЕНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ И ПЕРЕНОСОМ?

8.9. Укажите, какое из нижеперечисленных действий происходит при выполнении сложения с повышенной точностью с переносом, являющимся результатом предшествующей операции:

а) записывается в старший байт суммы;

б) теряется, если не записывается в память;

в) записывается в старший байт первого слагаемого;

г) записывается в разряд переноса регистра состояния.

8.10. Какое назначение схемы распределения памяти:

а) представление блок-схемы программы;

б) наглядное изображение использования памяти программой;

в) наглядное изображение последовательности выполнения программы?

8.11. При выполнении вычитания практически осуществляется сложение с

а) дополнительным кодом вычитаемого;

б) обратным кодом вычитаемого;

в) заемом?

8.12. Укажите, вычитание каких из нижеуказанных объектов реализует команда ВЫЧИТАНИЕ:

а) адресованных данных из заема;

б) адресованных данных из содержимого аккумулятора;

в) содержимого аккумулятора из адресованных данных;

г) адресованных данных из значения разряда переноса (заема) регистра состояния.

8.13. Какой из разрядов регистра состояния [а) нулевого результата, б) переноса, в) отрицательного результата, г) все указанные разряды] устанавливается в 1 или 0 в соответствии с результатом выполнения команды ВЫЧИТАНИЕ?

8.14. При выполнении команды ВЫЧИТАНИЕ С ПЕРЕНОСОМ осуществляется вычитание данных. Какое из следующих действий производится наряду с этим:

а) вычитание значения бита переноса;

б) обязательное формирование бита переноса;

в) прибавление значения бита переноса?

8.15. Какую из указанных ниже возможностей предоставляет команда ДЕСЯТИЧНАЯ КОРРЕКЦИЯ:

а) работать с десятичными числами;

б) корректировать результаты арифметических операций над двоично-десятичными числами;

в) корректировать результат выполнения двоичного сложения двоично-десятичных чисел;

г) выполнять деление двоично-десятичных чисел?

8.16. Отметьте, используются ли двоично-десятичные данные:

а) во всех устройствах управления печами СВЧ;

б) во всех терминалах торговых точек;

в) во многих простых изделиях;

г) для управления 7-сегментными индикаторами?

8.17. Сколько двоично-десятичных цифр размещается в каждом байте при использовании упакованных двоично-десятичных данных?

8.18. Какие из нижеперечисленных действий необходимо произвести при выполнении вычитания над двоично-десятичными числами:

а) использовать подпрограмму обработки двоично-десятичных данных;

б) использовать подпрограмму обработки данных в дополнительном коде;

в) учитывать значение разряда заема (переноса) регистра состояния;

г) предварительно преобразовать двоично-десятичные числа в десятичные?

8.19. Какое из приведенных здесь действий выполняется по команде ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ:

а) прибавляется значение разряда переноса к содержимому регистра;

б) вычитается двоичная 1 из содержимого регистра;

в) прибавляется двоичная 1 к содержимому регистра;

г) производится очистка регистра?

8.20. Для реализации какого устройства может быть использована как команда ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ, так и команда ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ:

а) программного счетчика, значение которого возрастает;

б) программного счетчика, значение которого уменьшается;

в) программного таймера?

8.21. Изобразите блок-схему программы сложения чисел с повышенной точностью.

8.22. Почему использование команды СЛОЖЕНИЕ С ПЕРЕНОСОМ представляется необходимым для реализации умножения чисел с повышенной точностью?

8.23. Объясните, в чем заключается различие воздействия результатов операций СЛОЖЕНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ и СЛОЖЕНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ И ПЕРЕНОСОМ на значение разряда переноса регистра состояния.

8.24. Объясните причины возникновения заема.

8.25. Какую информацию содержит схема распределения памяти?

8,26. а) В регистровой паре BC содержится число FFFF, которое подвергается положительному приращению. Каковы последствия этого действия?

б) В регистровой паре BC содержится число FFFF; производится положительное приращение содержимого регистра C. Каковы последствия этого действия?

Ответы на вопросы заданий для самопроверки

1. См. рис. 8.25. Наличие команды ЗАПИСЬ В ПАМЯТЬ ПРЯМАЯ увеличивает время выполнения программы на 5 мкс.

2. Первая команда LDA A заменяется на команду LDD A0008. С ее помощью в аккумулятор загружаются данные из области памяти 0008. Вторая команда ADI заменяется на команду ADD M0009. При ее выполнении содержимое области памяти 0009 складывается с содержимым аккумулятора. В качестве третьей команды применяется STA A,000A. Эта программа отличается от рассмотренной в качестве примера тем, что в ней используется только прямая адресация.

3.
```
LDA B,29
LDA A,0C
ADD B
HLT
```

Основное отличие данной программы от той, которая приведена в ответе на вопрос п. 2 заданий, состоит в использовании команды LDA B,29 для загрузки числа 29_{16} в регистр B.

Рис. 8.25. Блок-схема программы к ответу на вопрос п. 1 заданий для самопроверки.

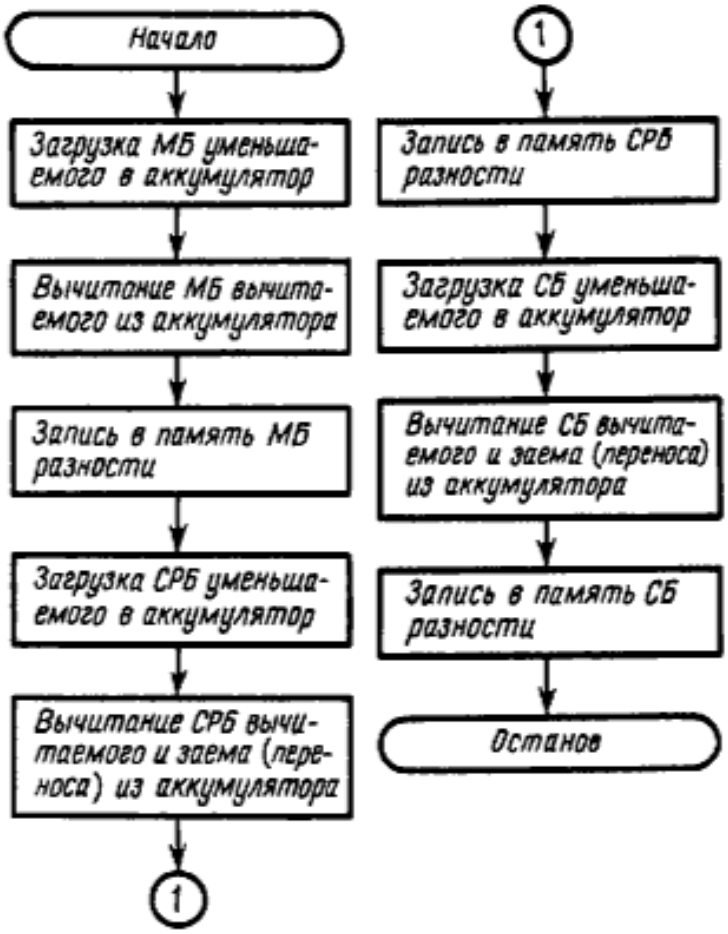


Рис. 8.26. Блок-схема программы к ответу на вопрос п. 10 заданий для самопроверки. (СРБ – средний байт.)

4. в.

5. В выполнении первой из этих команд сложения мог бы участвовать перенос, выработанный в результате выполнения предыдущей операции. В выполнении второй команды сложения перенос бы не участвовал совсем.

6. Во-первых, пришлось бы добавить по одной области памяти размером в байт для размещения обоих слагаемых и суммы. Во-вторых, потребовалось бы еще раз использовать фрагмент программы, состоящий из команд LDA A, ACD M и STA A, чтобы сложить третьи байты слагаемых.

7. Ни время выполнения программы, ни требуемый объем памяти не изменяются.

8. При выполнении этой команды значение бита разряда переноса используется как одно из исходных данных. В результате ее выполнения присваиваются значения всем разрядам регистра состояния.

9. При выполнении первой команды сложения исключается использование случайных значений разряда переноса. Последующие команды выполняются с участием ранее выработанных значений разряда переноса.

10. См. рис. 8.26.

11. Разряд переноса устанавливается в 1, когда

при вычитании требуется заем. Это происходит, когда вычитаемое оказывается больше уменьшаемого. Например:

Заем →	10011	3
	− 0101	− 5
Заем →	11110	−2

12. Разряд нулевого результата устанавливается в 1, когда все разряды результата имеют значение 0. Например:

1101	13
− 1101	− 13
0000	0

13. Разряд отрицательного результата устанавливается в 1, когда старший бит результата равен 1. Например:

0000 1111	15
− 0001 0000	− 16
1 1111 1111	−1
↑ ↑	↑
Заем Знак минус	Результат

14. Сформировать обратный код и прибавить к нему 1. Необходимо, кроме того, сохранить где-то информацию о том, что результат отрицательный, чтобы вывести на индикатор знак минус.
15. Потому что при выполнении команды ДЕСЯТИЧНАЯ КОРРЕКЦИЯ обязательно учитывается перенос, выработанный предшествующими командами.
16. Вы можете, как и ранее, использовать команду ДЕСЯТИЧНАЯ КОРРЕКЦИЯ при сложении, однако при вычитании необходимо прибегнуть к специальной подпрограмме.
17. 3) Младшая двоично-десятичная цифра результата меньше или равна 1001_2, а старшая превышает 1001_2; для коррекции прибавляется число 0110 0000. 4) Младшая и старшая двоично-десятичные цифры результата меньше или равны 1001_2; в коррекции нет необходимости.
18. См. рис. 8.27.
19. Потребуется 20-разрядное двоичное число, с помощью которого можно представить десятичные числа в диапазоне от 0 до 1 048 576. Количество байтов, необходимое для изображения слагаемых в виде двоичных чисел без знака и двоично-десятичных чисел, одинаково и равно 4.
20. Диапазон представления данных в виде 16-разрядных (двойного формата) двоичных чисел составляет ±2 147 483 648 или от 0 до 4 294 967 296. Упакованные двоично-десятичные данные, изображаемые посредством 16-раз-

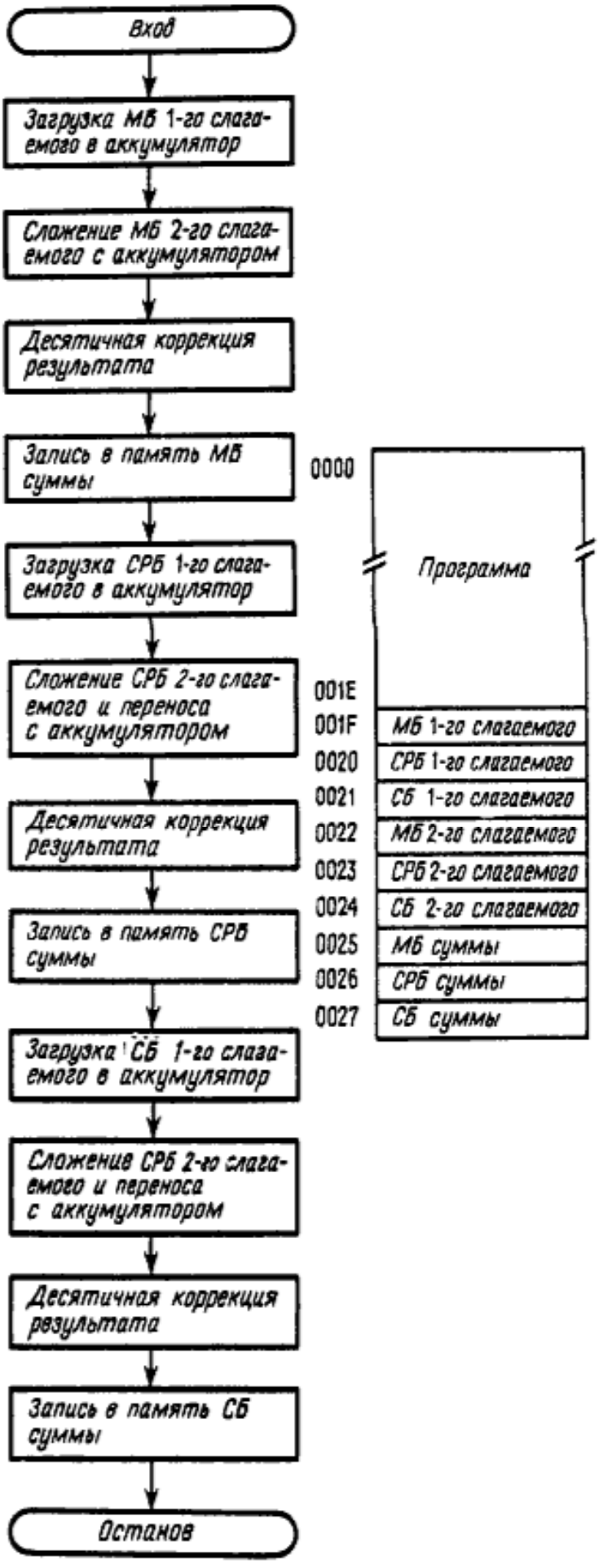


Рис. 8.27. Блок-схема программы и схема распределения памяти к ответу на вопрос п. 18 заданий для самопроверки.

рядных двоичных чисел, могут иметь величину от 0 до 99 999 999.

21.

	Содержимое регистра	Z	N	C
а)	1001 1000	0	1	0
б)	0000 0000	1	0	0
в)	0000 0001	0	0	0
г)	0001 0000	0	0	0
д)	1111 0001	0	1	0
е)	0000 0000 0000 0000	1	0	0
ж)	0000 0000 0000 0001	0	0	0
з)	0000 0001 0000 0000	0	0	0
и)	1111 1111 0000 0001	0	1	0

22.

	Содержимое регистра	Z	N	C
а)	1001 0110	0	1	0
б)	1111 1110	0	1	0
в)	1111 1111	0	1	0
г)	0000 1110	0	0	0
д)	1110 1111	0	1	0
е)	1111 1111 1111 1110	0	1	0
ж)	1111 1111 1111 1111	0	1	0
з)	0000 0000 1111 1110	0	0	0
и)	1111 1110 1111 1111	0	1	0

23. При поступлении тактового сигнала осуществляется положительное приращение содержимого счетчика. Точность часов определяется характеристиками генератора тактовых импульсов вычислительной машины.

24. Регистр с возможностью отрицательного приращения содержимого можно использовать для подсчета количества прибавлений множимого к содержимому аккумулятора. Блок-схема соответствующей программы приведена на рис. 8.28. Основной недостаток данного подхода – значительное время выполнения программы. Так, чтобы умножить 12 на 200, необходимо выполнить сложение 200 раз.

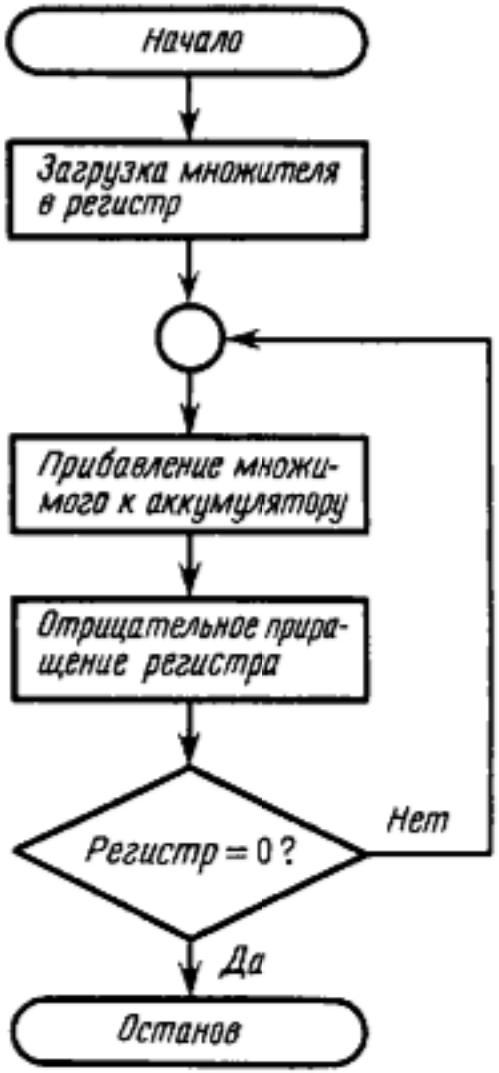


Рис. 8.28. Блок-схема программы к ответу на вопрос п. 24 заданий для самопроверки.

Глава 9. Логические команды

В этой главе мы ознакомимся с несколькими командами, обладающими большими возможностями. Как и арифметические команды, рассматриваемые здесь команды позволяют осуществлять обработку данных, находящихся в аккумуляторе микропроцессора. Однако, как следует из названия главы, они выполняют не арифметические, а логические операции.

В первой части главы будут рассмотрены четыре команды, с помощью которых могут быть реализованы основные логические операции: И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ и ОТРИЦАНИЕ (ИНВЕРСИЯ). Описана также команда СРАВНЕНИЕ, которая является основной при осуществлении различных проверок. Оказывается, такие проверки необходимы при постановке задач, в ходе выполнения которых требуется принимать решения.

Во второй части главы рассмотрены команды простого и циклического сдвигов. Вы увидите, как выполняются и в чем различны команды этих двух типов, а также научитесь совместно применять эти команды для сдвига информации увеличенного формата.

9.1. Команды И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ и ИНВЕРСИЯ

В воображении читателя микропроцессорная система прежде всего ассоциируется с системой «перемолачивания чисел», т.е. с решением сложных арифметических задач. Однако многие применения микропроцессорных устройств предполагают не арифметическую, а логическую обработку информации. Микро-ЭВМ может заменить собой множество логических схем.

В качестве примера рассмотрим систему на основе однокристальной микро-ЭВМ, изображенную на рис. 9.1. Система выполняет функцию дистанционного управления индикатором. Так как это единственная задача, реализуемая данной микропроцессорной системой, ее можно отнести к классу специализированных задач.

Показанная на рис. 9.1 микро-ЭВМ принимает данные, поступающие в нее по каналу последовательной передачи. При таком способе передачи данные поступают в микропроцессор последовательно, бит за битом. Микропроцессор преобразует их в параллельную форму с целью вывода на четыре 7-сегментных индикатора. Кроме того, микропроцессор осуществляет последовательное включение индикаторов. Это означает, что он включает сначала индикатор 1, затем индикатор 2, потом индикатор 3 и, наконец, индикатор 4, причем делает это настолько быстро, что мы не замечаем, как индикаторы включаются и гаснут.

Те же действия могут быть выполнены с помощью значительного количества интегральных схем ТТЛ малого и среднего уровней интеграции. Однако индикатор на основе однокристальной микро-ЭВМ является более дешевым устройством. В рассматриваемом примере микропроцессорная система по существу заменяет собой набор интегральных схем. Сложной обработки данных

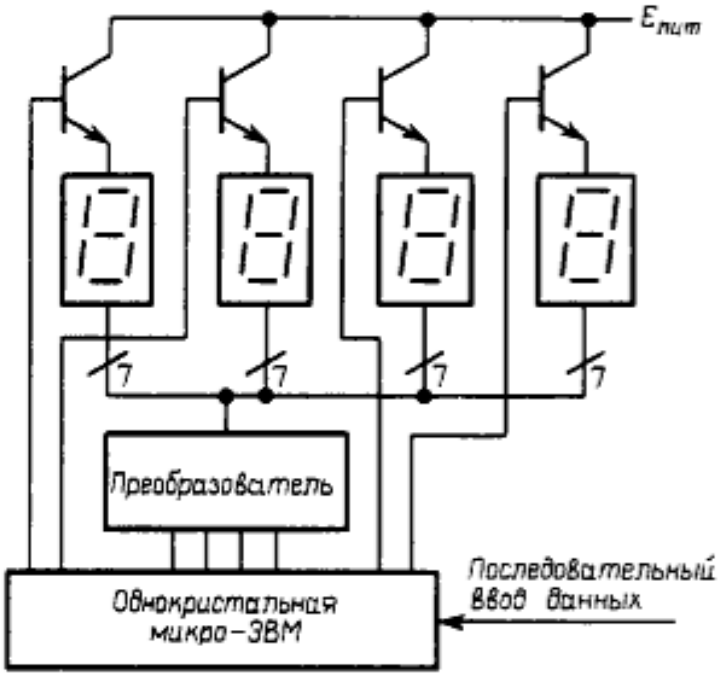


Рис. 9.1. Система с однокристальной микро-ЭВМ, заменяющей собой множество логических схем.

микропроцессор в данном случае не осуществляет.

Правила формирования результатов логических операций необходимо сформулировать в виде логических выражений. Как правило, это предполагает выполнение логических операций над обрабатываемыми данными. Например, над данными может быть произведена операция И, операция ИЛИ; данные могут быть также обработаны с использованием логических операций ИСКЛЮЧАЮЩЕЕ ИЛИ и ОТРИЦАНИЕ.

Все эти операции могут быть выполнены рассмотренным выше микропроцессором. Конечно, при реализации каждой логической функции могут быть использованы различные виды адресации. В данном разделе мы ознакомимся с возможными разновидностями логических функций. Ниже будут рассмотрены соответствующие типы команд, а также примеры, иллюстрирующие их выполнение.

На рис. 9.2 приведены таблицы истинности логических функций И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ и ОТРИЦАНИЕ. (При чтении выражений не путайте логическую операцию ИЛИ и арифметическую операцию СЛОЖЕНИЕ, для обозначения которых служит один и тот же символ +.)

Все логические операции являются побитовыми. Можно считать, что для обработки битов каждого разряда используется одна двухвходовая логическая схема. На один из ее входов подается значение бита аккумулятора, а на другой – значение соответствующего бита слова, расположенного в области памяти или регистре. В восьмиразрядном микропроцессоре предусмотрено восемь таких двухвходовых схем – по одной для каждого разряда слова микропроцессора.

Рис. 9.3 дает представление о процессах, имеющих место в ходе выполнения операции И. На практике для реализации логических операций служит арифметическо-логическое устройство микропроцессора. (*Примечание.* Не забывайте, что принят порядок нумерации битов от 0-го до 7-го справа налево!)

Как было замечено, логические операции выполняются всегда над содержимым аккумулятора и каким-то другим словом из регистра или из памяти. По окончании логической операции результат загружается в аккумулятор. Если результат равен 0 или если его старший разряд равен 1, то производится установка в 1 соответствующего бита регистра состояния. Значение бита переноса от результата выполнения логических команд И, ИЛИ и ИСКЛЮЧАЮЩЕЕ ИЛИ не зависит. Это означает, что исходное содержимое аккумулятора и данного бита состояния теряются.

Команда И (AND) может быть следующих четырех типов:

И над регистром и аккумулятором

AND r $A \cdot r \rightarrow A$

AND r

1 байт, два цикла

И над непосредственно адресуемой памятью и аккумулятором

AND M, Адрес $A \cdot M \rightarrow A$

AND M
СБ адреса
МБ адреса

3 байт, четыре цикла

И

Входные сигналы		Выходной сигнал
Бит аккумулятора	Бит памяти или регистра	Результирующий бит аккумулятора
0	0	0
0	1	0
1	0	0
1	1	1

ИЛИ

Входные сигналы		Выходной сигнал
Бит аккумулятора	Бит памяти или регистра	Результирующий бит аккумулятора
0	0	0
0	1	1
1	0	1
1	1	1

ИСКЛЮЧАЮЩЕЕ ИЛИ

Входные сигналы		Выходной сигнал
Бит аккумулятора	Бит памяти или регистра	Результирующий бит аккумулятора
0	0	0
0	1	1
1	0	1
1	1	0

ОТРИЦАНИЕ

Входные сигналы	Выходной сигнал
Бит аккумулятора	Результирующий бит аккумулятора
0	1
1	0

Рис. 9.2. Таблицы истинности операций И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ и ОТРИЦАНИЕ, используемых при реализации логических функций микропроцессором.

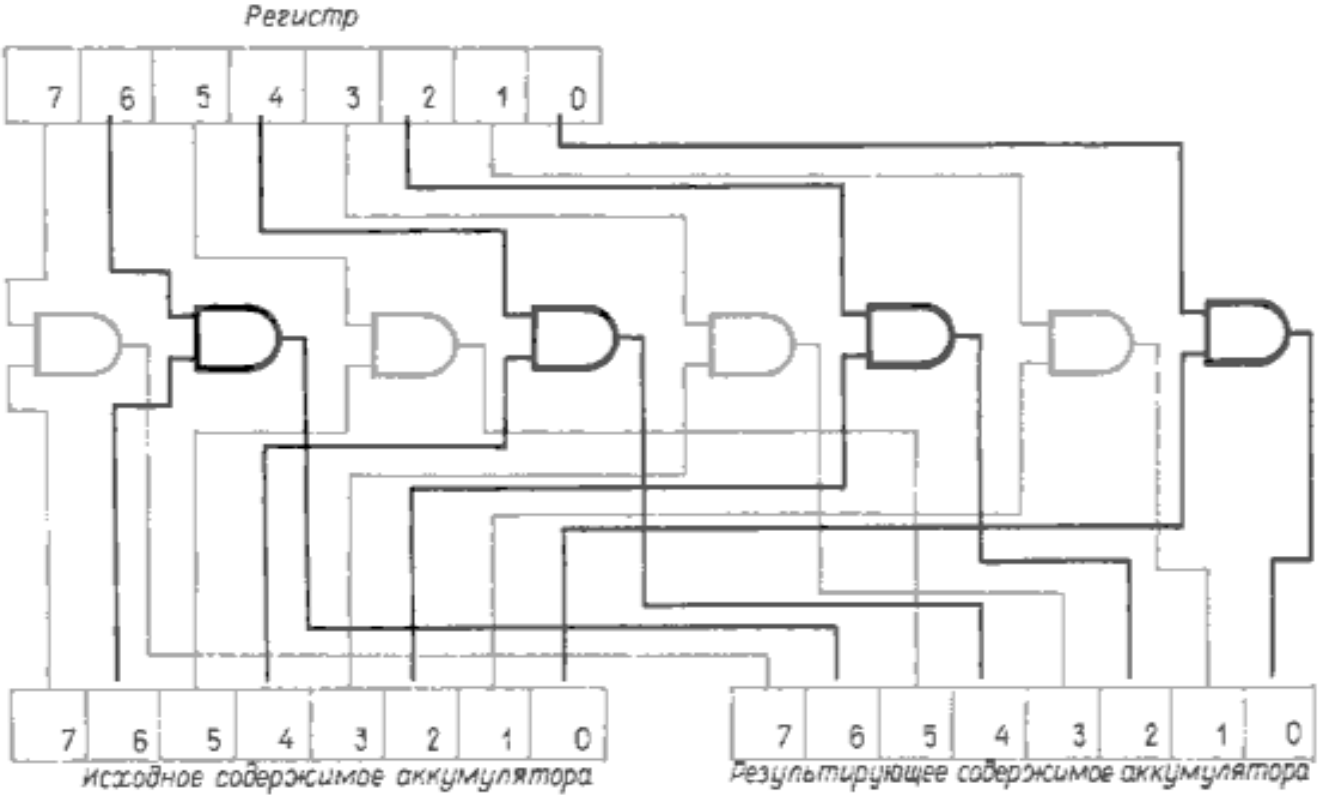
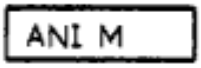


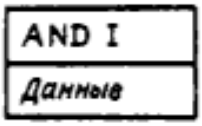
Рис. 9.3. Логическая схема, иллюстрирующая выполнение команды И.

И над косвенно адресуемой памятью и аккумулятором
ANI M $A \cdot M \rightarrow A$



1 байт, три цикла

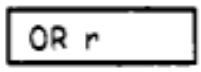
И над непосредственными данными и аккумулятором
AND I,Данные $A \cdot Данные \rightarrow A$



2 байт, три цикла

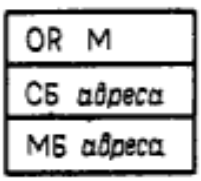
Команда ИЛИ бывает четырех типов:

ИЛИ над регистром и аккумулятором
OR r $A + r \rightarrow A$



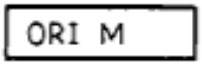
1 байт, два цикла

ИЛИ над непосредственно адресуемой памятью и аккумулятором
OR M,Адрес $A + M \rightarrow A$



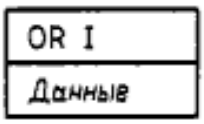
3 байт, четыре цикла

ИЛИ над косвенно адресуемой памятью и аккумулятором
ORI M $A + M \rightarrow A$



1 байт, три цикла

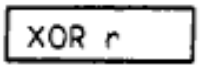
ИЛИ над непосредственными данными и аккумулятором
OR I,Данные $A + Данные \rightarrow A$



2 байт, три цикла

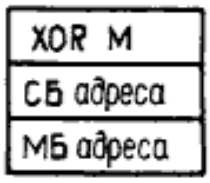
Команда ИСКЛЮЧАЮЩЕЕ ИЛИ имеет следующие типы:

ИСКЛЮЧАЮЩЕЕ ИЛИ над регистром и аккумулятором
XOR r $A \oplus r \rightarrow A$



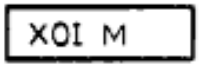
1 байт, два цикла

ИСКЛЮЧАЮЩЕЕ ИЛИ над непосредственно адресуемой памятью и аккумулятором
XOR M,Адрес $A \oplus M \rightarrow A$



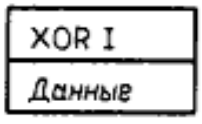
3 байт, четыре цикла

ИСКЛЮЧАЮЩЕЕ ИЛИ над косвенно адресуемой памятью и аккумулятором
XOI M $A \oplus M \rightarrow A$



1 байт, три цикла

ИСКЛЮЧАЮЩЕЕ ИЛИ над непосредственными данными и аккумулятором
XOR I,Данные $A \oplus Данные \rightarrow A$



2 байт, три цикла

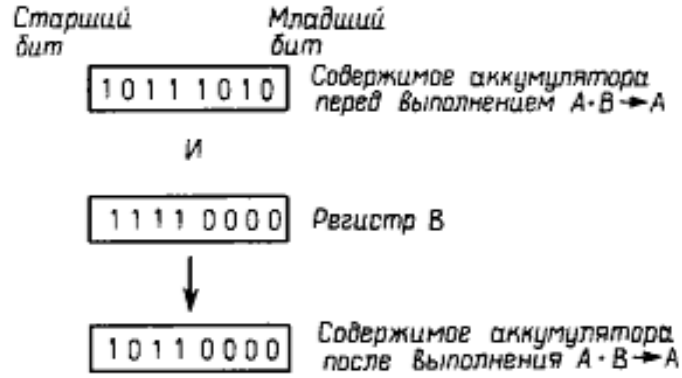


Рис. 9.4. Выполнение логической операции И над содержимым аккумулятора и содержимым регистра B.

При использовании логических команд необходимо помнить, что логические операции выполняются поразрядно. Пример реализации команды AND B показан на рис. 9.4.

В данном случае старший разряд результата представляет собой логическую 1. Это является следствием того, что как старший бит аккумулятора, так и старший бит регистра B равны 1. В то же время следующий бит результата (бит 6) имеет значение логического 0, потому что лишь один из соответствующих битов аккумулятора (регистра A) и регистра B имеет значение логической 1.

Пример, показанный на рис. 9.4, иллюстрирует, кроме того, одно из типичных применений команды И, называемое обычно *поразрядным маскированием*. Предположим, что нужно проверить наличие логических 0 в старших разрядах аккумулятора. При этом необходимо обеспечить, чтобы значения младших битов аккумулятора на результатах проверки не сказывались. Представляя себе особенности источника данных, мы знаем, что эти четыре младших бита могут содержать любую комбинацию от 0000 до 1111. Поэтому перед выполнением проверки необходимо каким-то образом исключить эти младшие биты из рассмотрения.

Чтобы узнать, не равны ли нулю значения всех восьми разрядов аккумулятора, анализируется значение разряда нулевого результата в регистре состояния. Как нам известно, этот разряд устанавливается в 1 в том и только в том случае, если значения всех разрядов аккумулятора представляют собой логические 0.

Прежде чем выяснить значение разряда нулевого результата в регистре состояния, необходимо выполнить операцию И над содержимым аккумулятора и набором битов в регистре B. Набор битов, имеющий место в регистре B, позволяет исключить из проверки содержимого аккумулятора четыре его младших разряда.

Как следует из рисунка, выполнение операции И не изменило значения четырех старших битов аккумулятора. Однако четырем младшим битам присвоено значение логического 0, потому что такое значение имеют соответствующие биты регистра B. Описанная операция называется *маскированием* четырех младших разрядов аккумулятора.

В качестве маски может быть использован любой набор битов. Могут быть, например, маскированы каждый второй разряд, четыре средних или три младших разряда. Так как при выполнении команды И применяются различные способы адресации, можно либо записать маску в память и затем обращаться к ней прямым или косвенным образом, либо загрузить ее в какой-либо другой регистр, либо представить маску в виде второго байта команды И над непосредственными данными. Пожалуй, наиболее часто маска имеет вид байта непосредственных данных.

Рис. 9.5 иллюстрирует выполнение логической команды ИЛИ. Бит результата, размещаемого в аккумуляторе, равен 1, если

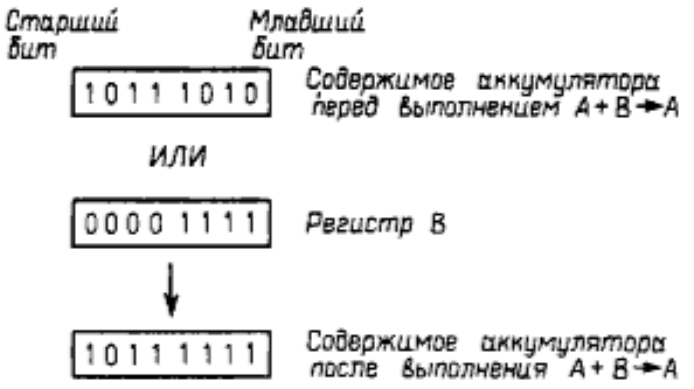


Рис. 9.5. Выполнение логической операции ИЛИ над содержимым аккумулятора и содержимым регистра B.

такое значение имеют соответствующие биты исходного содержимого аккумулятора или второго операнда, в качестве которого выступает либо содержимое области памяти, либо какого-то регистра.

Как можно видеть из рис. 9.5, команда ИЛИ также может быть использована для поразрядного маскирования. Если бит маски равен 0, то соответствующий бит содержимого аккумулятора остается без изменений. В тех случаях, когда биты маски равны 1, значения соответствующих битов исходного содержимого аккумулятора «блокируются», т.е. эти биты принимают единичные значения. В зависимости от цели, которую мы стремимся достичь, следует пользоваться либо маскированием по ИЛИ, либо маскированием по И.

Пример операции ИСКЛЮЧАЮЩЕЕ ИЛИ показан на рис. 9.6. Биты результата, размещаемого в аккумуляторе, имеют значение 0, если значения соответствующих битов исходного содержимого аккумулятора и слова данных совпадают, т.е. оба равны либо логической 1, либо логическому 0.

В данном примере выявляется совпадение значений битов 1, 2, 6, 7, а также отсутствие совпадения битов 0, 3, 4 и 5.

Приведенный пример иллюстрирует один из распространенных способов применения операции ИСКЛЮЧАЮЩЕЕ ИЛИ – ее использование для осуществления проверок. Бит проверяемых исходных данных обращается в логический 0, если он совпадает по значению с соответствующим битом маски. После выполнения операции

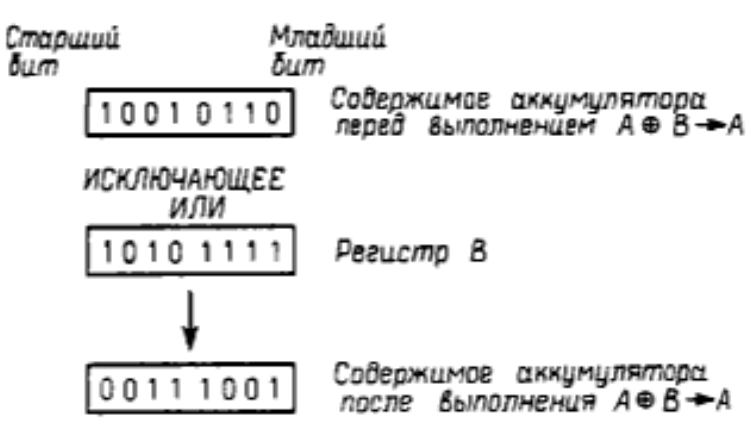


Рис. 9.6. Выполнение логической операции ИСКЛЮЧАЮЩЕЕ ИЛИ над содержимым аккумулятора и содержимым регистра B.

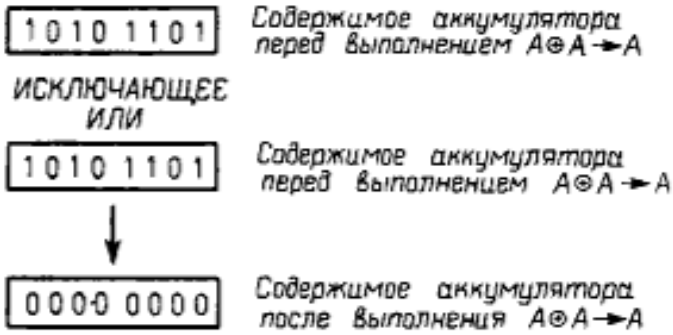


Рис. 9.7. Применение операции ИСКЛЮЧАЮЩЕЕ ИЛИ для очистки аккумулятора.

ИСКЛЮЧАЮЩЕЕ ИЛИ можно проверить значение каждого бита результата. Проверка осуществляется путем сдвига проверяемого бита в позицию старшего разряда аккумулятора и последующего выяснения значения разряда отрицательного результата в регистре состояния. Равенство нулю разряда отрицательного результата означает, что старший бит аккумулятора равен 0, а значит, проверяемый бит данных совпадает по значению с соответствующим битом маски.

Эта простая проверка применяется для ввода ряда независимых битов информации через один порт параллельного ввода-вывода и последующего выяснения значения каждого из этих битов. Биты, не удовлетворяющие условиям проверки (т.е. не совпадающие по значению с соответствующими битами маски), сразу же выявляются, так как им присваивается значение логической 1.

Другое применение команды ИСКЛЮЧАЮЩЕЕ ИЛИ состоит в очистке слова данных. У некоторых микропроцессоров нет специальной команды сброса, вместо нее можно прибегнуть к команде ИСКЛЮЧАЮЩЕЕ ИЛИ. Для осуществления сброса следует выполнить данную команду над словом, битам которого требуется присвоить нулевое значение, и копией этого слова. Происходящее при этом действие наглядно иллюстрирует рис. 9.7 – результат выполнения операции ИСКЛЮЧАЮЩЕЕ ИЛИ над битом и его копией представляет собой логический 0. Таким образом, при выполнении операции ИСКЛЮЧАЮЩЕЕ ИЛИ над любым словом данных и его копией производится *сброс*, или *очистка*, слова.

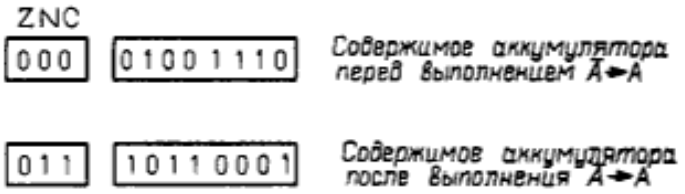
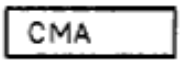


Рис. 9.8. Выполнение операции отрицания, которая сопровождается соответствующей установкой разрядов отрицательного результата и переноса в регистре состояния.

При выполнении команды ИНВЕРСИЯ над аккумулятором осуществляется поразрядное инвертирование содержимого аккумулятора, т.е. каждый бит, имевший значение логической 1, принимает значение логического 0, и наоборот. Для реализации команды ИНВЕРСИЯ используется логическая операция отрицания; при этом формируется *обратный код* (дополнение до 1) слова данных. На рис. 9.8 показано содержимое аккумулятора до и после выполнения команды ИНВЕРСИЯ.

Часто возникает необходимость получения *дополнительного кода* (дополнения до 2). Его можно сформировать путем образования обратного кода и последующего положительного приращения содержимого аккумулятора прибавления к нему 1.

ИНВЕРСИЯ АККУМУЛЯТОРА
CMA $\quad \bar{A} \rightarrow A$



При выполнении команды CMA разряды нулевого и отрицательного результатов устанавливаются в 1, если результат соответственно равен 0 или содержит 1 в старшем разряде. Разряд переноса всегда устанавливается в 1. Команда ИНВЕРСИЯ АККУМУЛЯТОРА имеет длину 1 байт и выполняется за два цикла.

Рассматриваемый ниже пример иллюстрирует применение этой команды. Известно, что микропроцессор выполняет арифметические команды над данными, представленными в одной из двух форм: в виде двоичных чисел без знака или в виде дополнительных кодов двоичных чисел со знаком. При изображении отрицательных чисел в дополнительном коде довольно просто реализуется операция вычитания. Для того чтобы выяснить значение такого числа, необходимо перевести его в прямой код. После такого преобразования отрицательного числа нужно каким-то образом указать, что оно является отрицательным. В большинстве индикаторов для этой цели имеется знак минус (–). Преобразование дополнительного кода отрицательного числа в двоичное число без знака осуществляется путем формирования обратного кода этого отрицательного числа и последующего прибавления 1. Указанные действия показаны на рис. 9.9.

На рис. 9.9, *а* приведен простой пример на выполнение вычитания, результатом которого является отрицательное число, так как вычитаемое больше уменьшаемого. О том, что результат представляет собой отрицательное число, можно судить по наличию 1 в его старшем разряде. Ответ представлен в дополнительном коде.

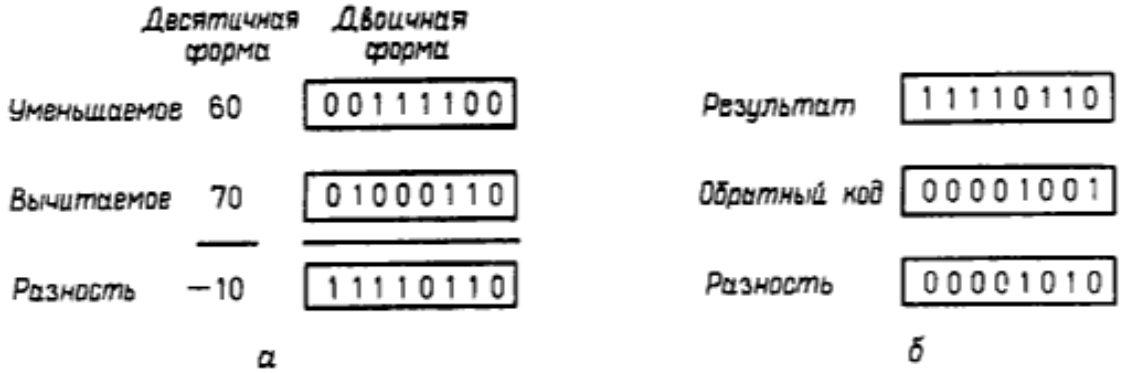


Рис. 9.9. Применение команд ИНВЕРСИЯ и ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ для выполнения операции формирования дополнительного кода. (Многие микропроцессоры располагают специальной командой, объединяющей функции, реализуемые этими двумя командами.)

Рис. 9.9, *б* иллюстрирует метод преобразования результата в двоичное число без знака. Для этого формируется обратный код результата, т. е. все имеющиеся в нем единицы заменяются нулями, и наоборот. В микропроцессоре это действие выполняется с помощью команды ИНВЕРСИЯ АККУМУЛЯТОРА. Чтобы получить результат в виде двоичного числа без знака, к содержимому аккумулятора прибавляется 1 с помощью команды ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ АККУМУЛЯТОРА. После образования обратного кода и прибавления к нему 1 видно, что абсолютное значение результата равно 10_{10}. Однако выполненная ранее проверка старшего разряда дополнительного кода результата показала, что последний имеет отрицательное значение. Двоичный результат может быть выведен на индикатор в виде десятичного числа 10. При этом должен быть включен также индикатор знака минус, т. е. на экране будет отображен истинный результат -10.

Команда ИНВЕРСИЯ АККУМУЛЯТОРА может следовать за командой И. При этом реализуется логическая операция И-НЕ. Точно так же выполнение команды ИНВЕРСИЯ АККУМУЛЯТОРА после команды ИЛИ позволяет реализовать операцию ИЛИ-НЕ, а после команды ИСКЛЮЧАЮЩЕЕ ИЛИ – операцию ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ.

Задания для самопроверки

1. Определите правильные результаты выполнения следующих операций:

а) 0011 1000
И
1010 1010

б) 1111 0000
ИЛИ
1010 1010

в) 0001 0000
ИСКЛЮЧАЮЩЕЕ ИЛИ
1111 0110

г) 0111 0000
ИЛИ
1010 0000

д) 1111 1111
И
1010 1010

е) 0000 0000
И
1111 1111

ж) 1111 1111
ИЛИ
1010 1010

з) 0000 0000
ИЛИ
1010 1010

и) 1111 1111
ИСКЛЮЧАЮЩЕЕ ИЛИ
1010 1010

к) 0000 0000
ИСКЛЮЧАЮЩЕЕ ИЛИ
1010 1010

2. Образуйте обратные коды следующих слов данных:

а) 0011 1000
б) 1111 0000
в) 0001 0000
г) 0111 0000
д) 1111 1111
е) 0000 0000
ж) 1010 1010
з) 0101 0101
и) 0000 1111

3. Операцию ИСКЛЮЧАЮЩЕЕ ИЛИ можно применить для проверки идентичности двух слов данных. Для этого надо выполнить операцию и проанализировать значение разряда нулевого результата регистра состояния. При этом выясняется, все ли разряды аккумулятора имеют значение логического 0. В чем заключается недостаток использования команды ИСКЛЮЧАЮЩЕЕ ИЛИ для выполнения описанной операции? (Этот недостаток настолько значительный, что многие микропроцессоры снабжены специальной командой, которая от него свободна.)

4. Команда ИСКЛЮЧАЮЩЕЕ ИЛИ может быть использована для установки всех разрядов аккумулятора в 0, если в качестве операндов служат содержимое аккумулятора и его копия. С помощью какой логической команды и над какими данными она должна выполняться, чтобы всем разрядам аккумулятора можно было присвоить значение 1?

5. Предположим, что нужно выполнить упаковку двоично-десятичных данных в 8-разрядном микропроцессоре. В аккумулятор введена и сдвинута в четыре старших разряда первая двоично-десятичная цифра – 1001. Содержимое аккумулятора имеет вид

1001 0000

Вторая двоично-десятичная цифра находится в регистре B:

0000 1000

Какую логическую операцию следует использовать для формирования из этих двух слов одного упакованного двоично-десятичного слова? Какой вид оно будет иметь после выполнения этой операции?

6. Исходное содержимое аккумулятора имело вид

1111 1000

Какие операции по обработке и пересылке этих данных должны быть выполнены для формирования содержимого регистра B, указанного в вопросе 5?

7. После команды И выполняется команда ИНВЕРСИЯ. Какова логическая связь результата реализации этой последовательности команд с исходными данными?

8. Изобразите блок-схему программы вычитания 1-байтовых чисел с последующим выводом результата со знаком на устройство отображения.

9.2. Команда СРАВНЕНИЕ

Часто возникает ситуация, когда требуется сравнить два двоичных числа. Обычно это связано с необходимостью выяснить, равны ли эти числа. Иногда нужно установить, какое из них больше или меньше.

Факт равенства двух чисел можно выявить с помощью команды ИСКЛЮЧАЮЩЕЕ ИЛИ. Можно, например, использовать для этой цели команду ИСКЛЮЧАЮЩЕЕ ИЛИ над непосредственными данными и аккумулятором. С помощью этой команды устанавливается, равны ли между собой данные, содержащиеся в аккумуляторе и во втором байте команды. Для осуществления проверки на равенство слово, равенство которого другому слову надо проверить, пересылается в аккумулятор. Затем выполняется команда ИСКЛЮЧАЮЩЕЕ ИЛИ над непосредственными данными и аккумулятором. Если исходное содержимое аккумулятора и непосредственное слово данных равны, результат в аккумуляторе содержит нули во всех разрядах. Чтобы узнать, так ли это, достаточно проверить значение разряда нулевого результата в регистре состояния.

Описанный способ сравнения двух слов имеет существенный недостаток. При выполнении сравнения теряется исходное содержимое аккумулятора. Если эти данные должны быть снова использованы, необходимо каким-то образом их запомнить перед выполнением команды ИСКЛЮЧАЮЩЕЕ ИЛИ. Для этой цели приходится вводить дополнительные команды.

Аналогичную проверку можно осуществить и с помощью команды ВЫЧИТАНИЕ. Эталонные данные вычитаются из данных, находящихся в аккумуляторе. Если эти два слова данных равны, устанавливается в 1 разряд нулевого результата в регистре состояния. Если значение данных в аккумуляторе (A) меньше, чем значение эталонных данных (R_D), т.е. $A < R_D$, то в 1 устанавливается разряд переноса, так как имеет место заем. Если значение данных в аккумуляторе больше значения эталонных данных, т.е. $A > R_D$, то ни один из разрядов регистра состояния не устанавливается в 1.

С помощью команды ВЫЧИТАНИЕ можно осуществлять не только проверку на равенство, но и проверки на «меньше» и «больше». Таким образом, команда ВЫЧИТАНИЕ предоставляет более широкие возможности для проведения сравнения. Однако сравнению с помощью команды ВЫЧИТАНИЕ присущ тот же недостаток, что и сравнению с помощью команды ИСКЛЮЧАЮЩЕЕ ИЛИ, т.е. потеря исходного содержимого аккумулятора при выполнении операции.

От этого недостатка свободна команда СРАВНЕНИЕ. Выполнение этой команды аналогично выполнению команды ВЫЧИТАНИЕ, однако ее результат не загружается в аккумулятор. Хотя по окончании операции исходные данные в аккумуляторе не изменяются, разрядам нулевого и отрицательного результата, а также разряду переноса присваиваются значения, соответствующие особенностям полученного результата.

В команде СРАВНЕНИЕ используются обычные способы адресации, т.е. с ее помощью можно сравнивать содержимое любого регистра и аккумулятора, непосредственно адресуемой области памяти и аккумулятора, косвенно адресуемой области памяти и аккумулятора, байта непосредственных данных и аккумулятора. Четыре типа этой команды приведены ниже:

СРАВНЕНИЕ регистра с аккумулятором
CMP r A – r

CMP r

1 байт, два цикла

СРАВНЕНИЕ непосредственно адресуемой памяти с аккумулятором
CMP M,Адрес A – M

CMP
СБ адреса
МБ адреса

3 байт, четыре цикла

СРАВНЕНИЕ косвенно адресуемой памяти с аккумулятором
CMI M A – M

CMI M

1 байт, два цикла

СРАВНЕНИЕ непосредственных данных с аккумулятором

CMP I,Данные A – Данные

CMP I
Данные

2 байт, три цикла

Еще раз напомним, что результат операции СРАВНЕНИЕ не отражается на содержимом аккумулятора, а лишь обусловливает соответствующую установку разрядов нулевого и отрицательного результатов и переноса.

Применение команды СРАВНЕНИЕ позволяет принимать решения в ходе выполнения программы. Если, например, проверкой выявляется равенство двух слов, то выполнение программы продолжается в одном направлении, а если слова оказались неравными, то в другом. В программе чтения файла данных, блок-схема которой приведена на рис. 9.10, *а*, использована команда СРАВНЕНИЕ.

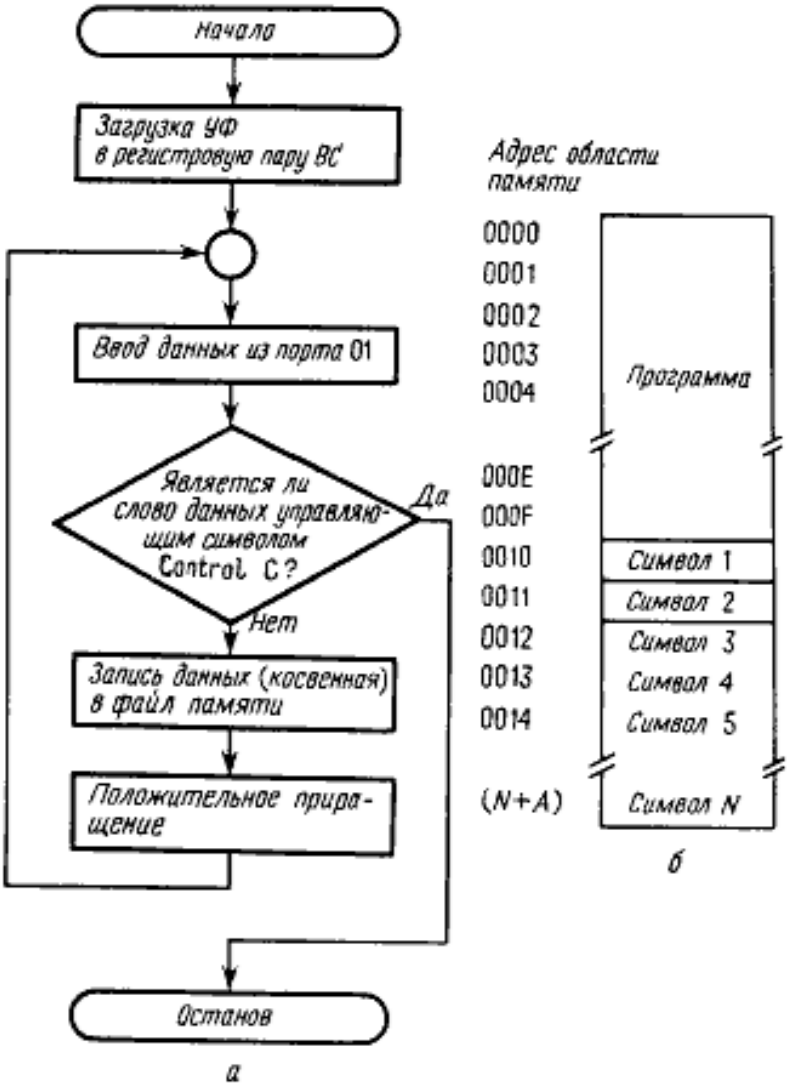


Рис. 9.10. *а* – Блок-схема программы для занесения символов кода ASCII в последовательные области памяти и проверки наличия управляющего символа Control C (УФ – указатель файла); *б* – схема распределения памяти.

Эта программа осуществляет ввод слов данных через порт ввода-вывода 01. Так как эти слова формируются с помощью клавиатуры с символами кода ASCII[1)], они являются 8-битовыми. Каждое из них содержит 7 бит данных и 1 бит проверки на четность, служащий для обнаружения ошибок в данных после их передачи. Каждое слово представляет собой отдельный буквенно-цифровой знак или специальный символ кода ASCII. В ходе нормального выполнения программы каждый такой символ записывается в очередную область памяти. Использование памяти иллюстри-

[1)] ASCII (American Standard Code for Information Interchange) – Стандартный американский код для обмена информацией. – *Прим. перев*

руется схемой распределения, приведенной на рис. 9.10, *б*.

Эта программа следит за появлением специального управляющего символа Control C кода ASCII. Появление этого символа означает окончание передачи данных. Для его обнаружения в данной программе используется команда СРАВНЕНИЕ. При появлении символа Control C выполнение программы прекращается. Полный листинг программы приведен на рис. 9.11.

Адрес области памяти	Содержимое области памяти	Комментарий
0000	LRP B	ЗАГРУЗКА РЕГИСТРОВОЙ ПАРЫ НЕПОСРЕДСТВЕННАЯ
0001	00	Адрес начала файла,
0002	10	загружаемый в BC
0003	IN	ВВОД данных из порта 01
0004	01	
0005	CMP I	СРАВНЕНИЕ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ
0006	03	Шестнадцатеричный код управляющего символа C
0007	JZ	ПЕРЕХОД, ЕСЛИ НУЛЬ (положительный результат сравнения)
0008	00	Местоположение в памяти
0009	0F	команды ОСТАНОВ
000A	STI A	ЗАПИСЬ В ПАМЯТЬ АККУМУЛЯТОРА КОСВЕННАЯ
000B	IRP	ПОЛОЖИТЕЛЬНОЕ ПРИРАЩЕНИЕ пары BC
000C	JMP	ПЕРЕХОД к началу цикла (команде ВВОД)
000D	00	Адрес команды ВВОД
000E	03	
000F	HLT	ОСТАНОВ
0010		Файл данных
0011		» »
0012		» »
0013		» »
0014		» »
0015		» »
0016		» »
0017		» »
0018		» »
0019		» »
001A		» »
001B		» »
001C		» »
001D		» »
001E		» »
001F		» »

Рис. 9.11. Листинг программы, блок-схема которой приведена на рис. 9.10.

Проанализируем ход выполнения программы команда за командой. Регистровая пара BC служит для указания на очередную свободную область памяти. Поэтому первая команда программы производит загрузку 16-разрядного адреса в эту пару регистров. Упомянутый адрес указывает на первую области памяти, в которой расположен файл данных. В рассматриваемом примере файл данных начинается с области памяти 0010.

По второй команде производится ввод данных из порта ввода-вывода 01, т.е. содержимое порта ввода-вывода пересылается в аккумулятор.

Пока слово данных находится в аккумуляторе, с помощью команды СРАВНЕНИЕ можно проверить, не является ли это слово кодом управляющего символа Control C. Делать это следует, не нарушая содержимого аккумулятора, так как если оно не представляет собой этого управляющего символа, то подлежит записи в память.

Символ Control C изображается 8-разрядным двоичным словом 00000011, т.е. шестнадцатеричным числом 03. Поэтому второй байт команды СРАВНЕНИЕ содержит шестнадцатеричное число 03. Если содержимое аккумулятора имеет именно такое значение, то команда СРАВНЕНИЕ устанавливает разряд нулевого результата регистра состояния в 1; если же содержимое аккумулятора представляет собой код какого-либо другого символа, то разряд нулевого результата остается в состоянии 0.

Выполнением четвертой команды (ПЕРЕХОД, ЕСЛИ НУЛЬ) управляет разряд нулевого результата регистра состояния. (Эта команда будет рассмотрена более подробно в гл. 10.) В рассматриваемой программе эта команда присваивает счетчику команд значение 000F, если установлен в 1 разряд нулевого результата в регистре состояния, т.е. если командой СРАВНЕНИЕ обнаружено наличие в аккумуляторе управляющего символа Control C. В этом случае выполнение программы передается команде, которая находится в области памяти по адресу 000F, т.е. команде ОСТАНОВ. Тем самым выполнение программы прекращается.

Если символ кода ASCII, находящийся в аккумуляторе, не является символом Control C, то разряд нулевого результата в состояние 1 не устанавливается, и, следовательно, перехода не осуществляется. Далее выполняется команда, хранимая в области памяти по адресу 000A, следующей за областью, где находится команда ПЕРЕХОД, ЕСЛИ НУЛЬ. Это – команда ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ. С ее помощью содержимое аккумулятора загружается в область памяти, на которую указывает регистровая пара BC. В данном случае при выполнении этой команды регистры BC указывают на область памяти 0010, потому что число 0010 было ранее занесено в регистровую пару BC с помощью команды ЗАГРУЗКА РЕГИСТРОВОЙ ПАРЫ.

После выполнения команды ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ производится положительное приращение содержимого регистровой пары BC, после чего она указывает на следующую область файла данных в памяти. Тем самым эта область подготавливается к приему второго слова входных данных.

Очередная команда является командой безусловного перехода. В соответствии с ней счетчик команд снова указывает на область 0003. Соответственно следующей выполняемой командой будет ВВОД 01. Описанный процесс ввода данных повторяется с начала.

Как можно видеть, выполнение программы продолжается до тех пор, пока не встретится символ Control C. Как только появляется этот символ, выполнение программы прекращается. Обычно подобные программы используются как составные части других программ, т.е. в качестве *подпрограмм*. Обращение к такой подпрограмме происходит каждый раз, когда возникает необходимость ввести в файл данных новый символ кода ASCII. Для ограничения размеров файла может применяться другая подпрограмма.

При обнаружении в аккумуляторе управляющего символа Control C с помощью команды СРАВНЕНИЕ не обязательно должен происходить останов программы; вместо этого может иметь место обращение к какой-либо другой подпрограмме.

Однако, какое бы действие ни выполнялось следующим, последовательность ввода файла остается в основном неизменной.

Задания для самопроверки

9. Предположим, что в коде ASCII имеется управляющий символ Control D, изображаемый в виде шестнадцатеричного числа 04.

а) Какие изменения следует внести в листинг программы, приведенный на рис. 9.11, чтобы останов произошел при обнаружении кода символа Control D, а не Control C? б) Как изменяется при этом блок-схема программы и схема распределения памяти (рис. 9.10)? в) Какие изменения необходимо внести в программу, чтобы останов происходил при обнаружении как кода символа Control C, так и кода символа Control D?

10. Объясните основные различия между командами СРАВНЕНИЕ и ИСКЛЮЧАЮЩЕЕ ИЛИ, а также между командами СРАВНЕНИЕ и ВЫЧИТАНИЕ.

11. Для слежения за изменением аналоговых данных часто применяются микропроцессоры. Микропроцессор должен выдать предупреждающий сигнал, если отслеживаемая величина превышает некоторый заданный уровень. Аналоговые данные преобразуются в цифровые слова в диапазоне от 00000000 до 01000000 с помощью аналого-цифрового преобразователя. Предположим, что предупреждающий сигнал необходимо выдавать, когда значение цифрового эквивалента аналоговой величины превышает величину 00100000. Каким образом можно это сделать с помощью команды СРАВНЕНИЕ? При объяснении используйте блок-схему программы.

12. Изобразите блок-схему программы, в которой команды сравнения и проверки значений разрядов регистра состояния используются для разветвления программы на три различных направления в соответствии с условиями «меньше», «равно» и «больше».

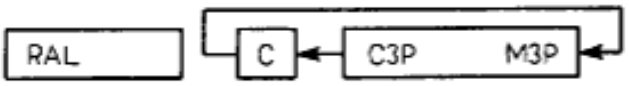
9.3. Команды простого и циклического сдвига

Следующие четыре команды, которые мы рассмотрим,–это команды СДВИГ и ЦИКЛИЧЕСКИЙ СДВИГ. В литературе по микропроцессорам в некоторых случаях все эти команды называют командами циклического сдвига, в других случаях – командами сдвига; встречается и совместное использование указанных терминов. Поэтому в каждом конкретном случае необходимо тщательно разобраться в материалах фирмы-изготовителя, чтобы точно установить, какие действия выполняются каждой из команд сдвига.

Зачастую в документации фирм, выпускающих микропроцессоры, приводится схематическое изображение выполнения команд; аналогичные схемы будут рассмотрены и в данном разделе.

Команды СДВИГ и ЦИКЛИЧЕСКИЙ СДВИГ 1-байтовые с двумя циклами. Если результат выполнения любой из них содержит 1 в старшем разряде или равен 0, то устанавливается в 1 соответствующий разряд регистра состояния. Разряд переноса участвует в выполнении этих операций, и его значение изменяется по мере передачи данных в него и из него.

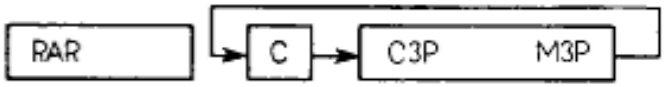
ЦИКЛИЧЕСКИЙ СДВИГ АККУМУЛЯТОРА ВЛЕВО
RAL



Командой ЦИКЛИЧЕСКИЙ СДВИГ АККУМУЛЯТОРА ВЛЕВО все данные, находящиеся в аккумуляторе и разряде переноса, сдвигаются на одну позицию влево, т.е. содержимое каждого разряда перемещается в позицию соседнего, старшего значащего разряда (СЗР). Содержимое старшего разряда аккумулятора перемещается в разряд переноса (С) регистра состояния. Содержимое разряда переноса регистра состояния перемещается в младший разряд (МР) аккумулятора.

Важно усвоить, что разряд переноса регистра состояния участвует в реализации команды. Содержимое старшего разряда аккумулятора не перемещается непосредственно в его младший разряд, а проходит через разряд переноса регистра состояния.

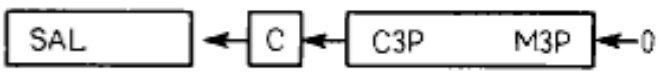
ЦИКЛИЧЕСКИЙ СДВИГ АККУМУЛЯТОРА ВПРАВО
RAR



По команде ЦИКЛИЧЕСКИЙ СДВИГ АККУМУЛЯТОРА ВПРАВО данные перемещаются в противоположном направлении. Содержимое младшего разряда аккумулятора перемещается в разряд переноса регистра состояния. Содержимое разряда переноса пересылается в старший разряд аккумулятора. Содержимое каждого разряда аккумулятора сдвигается в соседний, младший разряд. Подчеркнем еще раз, что сдвиг содержимого аккумулятора осуществляется с участием разряда переноса регистра состояния.

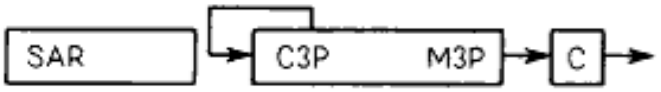
При использовании команд циклического сдвига не происходит потерь данных, поскольку данные перемещаются по замкнутому контуру. Так, например, если выполнить циклический сдвиг в 8-разрядном микропроцессоре девять раз, то окончательное содержимое аккумулятора и разряда переноса полностью совпадет с исходным.

СДВИГ АККУМУЛЯТОРА ВЛЕВО
SAL



Команду СДВИГ АККУМУЛЯТОРА ВЛЕВО называют также командой АРИФМЕТИЧЕСКИЙ СДВИГ ВЛЕВО. Она часто используется для реализации операции умножения. Как можно видеть из приведенной выше схемы, при выполнении этой команды содержимое старшего разряда аккумулятора передается в разряд переноса регистра состояния. В младший разряд аккумулятора загружается при этом 0. Содержимое каждого разряда аккумулятора перемещается в соседний, старший разряд, а содержимое разряда переноса (независимо от его значения) теряется.

СДВИГ АККУМУЛЯТОРА ВПРАВО
SAR



Команда СДВИГ АККУМУЛЯТОРА ВПРАВО аналогична команде СДВИГ АККУМУЛЯТОРА ВЛЕВО. Содержимое младшего разряда аккумулятора передается в разряд переноса регистра состояния. Содержимое всех остальных разрядов аккумулятора сдвигается вправо на одну позицию.

Отличительной особенностью команды СДВИГ АККУМУЛЯТОРА ВПРАВО является то, что старший разряд аккумулятора в процессе выполнения команды сохраняет свое значение. Вместо того чтобы записать в данный разряд логический 0, команда заносит в него то же значение, которое имело место до сдвига. Сохранение значения старшего разряда аккумулятора необходимо при выполнении некоторых арифметических операций в дополнительном коде.

Существуют два обстоятельства, которые необходимо иметь в виду, работая с командами простого и циклического сдвига. Во-первых, любая из выполняемых при этом операций эквивалентна умножению или делению числа на 2 (рис. 9.12 и 9.13).

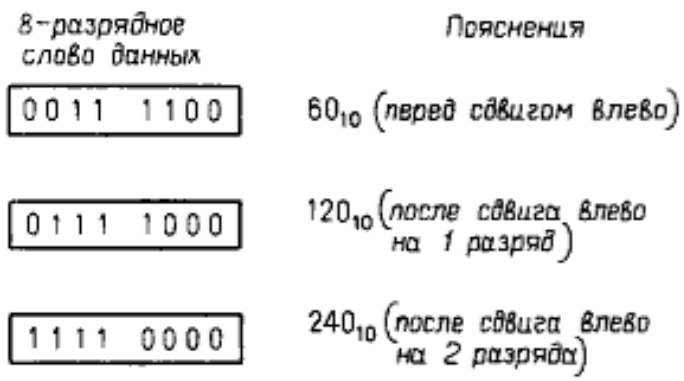


Рис. 9.12. Умножение на 2 путем сдвига влево.

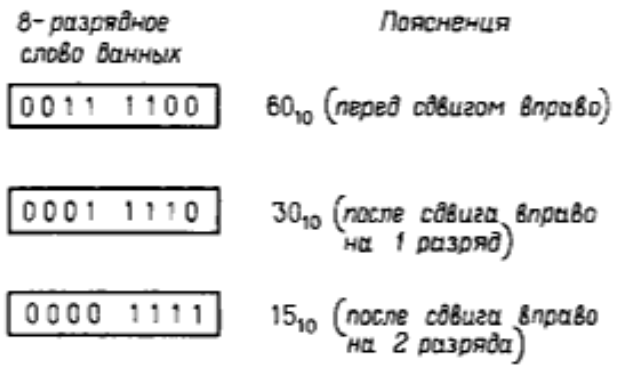


Рис. 9.13. Деление на 2 путем сдвига вправо.

На рис. 9.12 показано, что происходит при сдвиге влево. При сдвиге на одну позицию влево число, имевшее значение 60, удваивается, т.е. становится числом 120. В результате второго сдвига вновь происходит удвоение числа – теперь в слове находится число 240. Таким образом, выполнение каждой операции сдвига влево равносильно умножению на 2.

Рис. 9.13 иллюстрирует, какие действия происходят при сдвиге вправо. При сдвиге вправо на одну позицию вдвое уменьшается значение числа. Исходное число 60 преобразуется в число 30. В результате второго сдвига значение содержимого регистра вновь уменьшается в два раза, т.е. в слове содержится число 15. Очевидно, что выполнение каждой операции сдвига вправо равносильно делению числа на 2.

Второе обстоятельство, которое необходимо учитывать при использовании команд простого сдвига, заключается в том, что данные, сдвигаемые за пределы регистра, теряются. При выполнении команд циклического сдвига этого не происходит, так как в этом случае данные циркулируют по замкнутому контуру.

Рассмотрим пример, иллюстрирующий применение команд СДВИГ и ЦИКЛИЧЕСКИЙ СДВИГ как необходимых элементов выполнения операции умножения. Как известно, умножение двоичных чисел организуется как последовательность операций сдвига и сложения. Но что происходит, если двоичное число выходит за пределы диапазона 0–255? В соответствии с тем, что было установлено выше применительно к операциям сложения и вычитания, в таких случаях необходимо прибегать к представлению данных с повышенной точностью. Как и при сложении и вычитании, реализация умножения с повышенной точностью предполагает использование двух 8-разрядных регистров в качестве одного 16-разрядного. Значит, для осуществления умножения чисел с повышенной точностью необходимо иметь возможность производить сдвиг данных увеличенного формата.

Сдвиг данных тройного формата показан на рис. 9.14. На рис. 9.14, *а* изображены 3 байт данных, в которых содержится двоичное множимое. Как было выяснено ранее при изучении двоичной арифметики, для формирования частичных произведений множимое должно сдвигаться влево. Этот сдвиг должен выполняться побайтно. Младший байт сдвигается с помощью команды АРИФМЕТИЧЕСКИЙ СДВИГ ВЛЕВО. В ходе этого сдвига содержимое старшего бита этого байта заносится в разряд переноса регистра состояния, а в младший бит младшего байта помещается двоичный 0 (рис. 9.14, *б*).

Для сдвига второго байта команду АРИФМЕТИЧЕСКИЙ СДВИГ ВЛЕВО использовать нельзя, так как в его младший разряд надо переслать содержимое старшего бита первого байта. Поэтому в данном случае применяется команда ЦИКЛИЧЕСКИЙ СДВИГ АККУМУЛЯТОРА ВЛЕВО (рис. 9.14, *в*). По этой команде содержимое разряда переноса регистра состояния записывается в младший разряд аккумулятора, а содержимое старшего разряда аккумулятора передается в разряд переноса регистра состояния.

Третий, старший байт также сдвигается посредством команды ЦИКЛИЧЕСКИЙ СДВИГ АККУМУЛЯТОРА ВЛЕВО (рис. 9.14, *г*). Содержимое разряда переноса регистра состояния (образовавшееся при циклическом сдвиге среднего байта) посылается в младший разряд старшего байта. В разряд переноса регистра состояния передается из старшего бита данного байта логический 0.

Таким образом, с помощью одной операции арифметического сдвига и двух операций циклического сдвига содержимого аккумулятора влево можно осуществить сдвиг числа тройного формата на одну по-

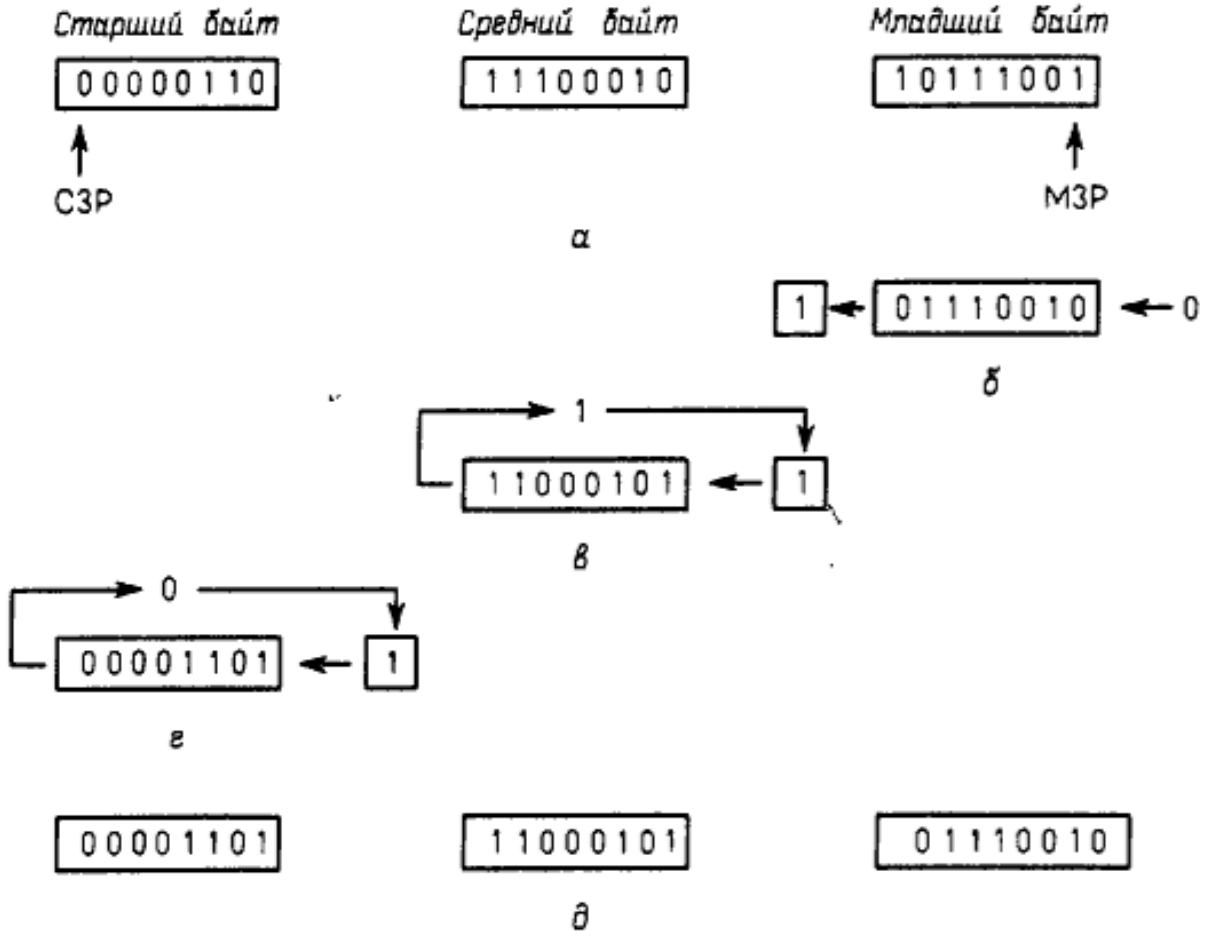


Рис. 9.14. Сдвиг влево на 1 разряд данных тройного формата.

зицию влево. Результат этого сдвига представлен на рис. 9.14, *д*.

В ходе выполнения перечисленных действий за один шаг происходит сдвиг только на одну позицию. При этом сначала должен быть произведен сдвиг младшего байта, затем байтов, следующих по старшинству. При необходимости сдвига данных еще на одну позицию все указанные операции следует повторить в той же последовательности.

Рассмотренный сдвиг данных тройного формата выполняется за один шаг лишь на одну позицию, так как разряд переноса регистра состояния представляет собой 1-разрядный регистр. Если попытаться осуществить сдвиг данных сразу на несколько позиций, то содержимое некоторых битов будет утеряно.

Задания для самопроверки

Изобразите в наглядной форме содержимое разряда переноса и аккумулятора после выполнения над данными, приведенными в пп. 13-16, следующих действий:

а) СДВИГ ВЛЕВО на один разряд;
б) СДВИГ ВЛЕВО на два разряда;
в) СДВИГ ВПРАВО на один разряд;
г) СДВИГ ВПРАВО на два разряда;
д) ЦИКЛИЧЕСКИЙ СДВИГ ВЛЕВО на один разряд;
е) ЦИКЛИЧЕСКИЙ СДВИГ ВЛЕВО на два разряда;
ж) ЦИКЛИЧЕСКИЙ СДВИГ ВПРАВО на один разряд;
з) ЦИКЛИЧЕСКИЙ СДВИГ ВПРАВО на два разряда.

	C	Аккумулятор
13.	0	1 1 0 1 0 1 0 0
14	0	0 0 1 0 1 1 1 1
15.	1	0 1 1 1 1 1 1 1
16.	0	1 1 1 1 1 1 1 0

17. Имеется микропроцессорная система, управляющая средствами поддержания определенных климатических условий

в здании. В микропроцессор можно ввести одно 8-разрядное слово данных, отражающее включенное и выключенное состояния этих средств. Логическая 1 означает, что соответствующее устройство включено, а логический 0 – что оно выключено. Назначение разрядов слова состояния объектов управления показано на следующем рисунке:

7	6	5	4	3	2	1	0
Кондиционер №1	Вентиль горячей воды №3	Кондиционер №2	Кондиционер №3	Вентиль горячей воды №2	Вентиль горячей воды №4	Кондиционер №4	Кондиционер №5

Разработайте блок-схему подпрограммы, обеспечивающей быстрый анализ приведенного слова состояния после загрузки его в аккумулятор. Задачей анализа является выяснение вопроса, включены ли какие-либо из имеющихся кондиционеров. Укажите, какая команда будет для этого использована, и значения специальных слов данных, которые окажутся необходимы.

Упражнения

9.1. С помощью каких команд (арифметических, пересылки, условного перехода, логических) реализуются булевы операции?

9.2. В выполнении каких из приведенных ниже команд участвует только содержимое аккумулятора:

а) И; б) ИЛИ; в) ИСКЛЮЧАЮЩЕЕ ИЛИ; г) НЕ?

9.3. Можно ли выполнить логические операции: а) над содержимым аккумулятора; б) над содержимым регистра, в котором находятся данные, являющиеся образцом для сравнения; в) поразрядно; г) побайтно?

9.4. Изменится ли при выполнении логических операций содержимое следующих разрядов регистра состояния: нулевого результата, отрицательного результата, переноса?

9.5. Загружается ли результат выполнения операций И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ и НЕ в регистр состояния, адресуемую область памяти, аккумулятор или же результат вообще не формируется?

9.6. Кратко объясните, что такое поразрядное маскирование.

9.7. Объясните различия в поразрядном маскировании, выполняемом с привлечением операций И и ИЛИ.

9.8. Какой результат загружается в аккумулятор при выполнении команды ИСКЛЮЧАЮЩЕЕ ИЛИ над непосредственными данными и аккумулятором, если изначально в аккумуляторе находилось число 38_{16}, а данными является число $C7_{16}$?

9.9. Что является результатом выполнения операции ИСКЛЮЧАЮЩЕЕ ИЛИ над словом данных и его копией?

9.10. Изобразите блок-схему программы преобразования числа 02DE в дополнительный код.

9.11. Образуйте обратный код следующих ниже слов данных. Укажите содержимое регистра состояния по окончании выполнения этих операций.

а) 0010 1100
б) 1111 1111
в) 1000 0000
г) 0000 0001
д) 0000 0000
е) 1100 1100
ж) 1010 1010
з) 0101 0101

9.12. В чем состоит различие между обратным и дополнительным кодами? Какие команды используются для прямого и обратного преобразований данных из одного кода в другой?

9.13. После операции ИСКЛЮЧАЮЩЕЕ ИЛИ выполняется операция отрицания. Битам с каким значением присваивается значение логической 1?

9.14. Какой из нижеследующих команд аналогична команда СРАВНЕНИЕ, за исключением того, что ее результат не отражается на содержимом аккумулятора:

а) СЛОЖЕНИЕ;
б) ВЫЧИТАНИЕ;
в) И;
г) ИСКЛЮЧАЮЩЕЕ ИЛИ?

9.15. Для какой из указанных ниже целей предназначена команда СРАВНЕНИЕ:

а) для проверки данных; в) для пересылки;
б) для записи в память; г) для сброса?

9.16. Команда СРАВНЕНИЕ над непосредственными данными служит для выявления: а) управляющего символа Control C; б) управляющего символа Control D; в) совпадения со-

держимого аккумулятора со вторым байтом команды или г) для выполнения всех названных действий?

9.17. Объясните действия, выполняемые при реализации команды CMP M,A3D1.

9.18. Какую команду следует использовать для выяснения того, содержится ли в регистре D число FA_{16}?

9.19. Объясните различия между командами ЦИКЛИЧЕСКИЙ СДВИГ и СДВИГ.

9.20. Объясните различия между командами простого сдвига влево и вправо и командами циклического сдвига влево и вправо.

9.21. Как изменяется содержимое разряда переноса при выполнении команды циклического сдвига?

9.22. Как изменяется содержимое разряда переноса при выполнении команды циклического сдвига?

9.22. Как изменяется содержимое разряда переноса при выполнении команды сдвига?

9.23. При выполнении каких команд простого и циклического сдвига остается без изменений содержимое старшего разряда аккумулятора? Какая цель преследуется при этом?

9.24. Что происходит с данными, перемещаемыми из разряда переноса при выполнении операции АРИФМЕТИЧЕСКИЙ СДВИГ?

Ответы на вопросы заданий для самопроверки

1. а) 0010 1000
   б) 1111 1010
   в) 1110 0110
   г) 1111 0000
   д) 1010 1010
   е) 0000 0000
   ж) 1111 1111
   з) 1010 1010
   и) 0101 0101
   к) 1010 1010

2. а) 1100 0111
   б) 0000 1111
   в) 1110 1111
   г) 1000 1111
   д) 0000 0000
   е) 1111 1111
   ж) 0101 0101
   з) 1010 1010
   и) 1111 0000

3. Данные, подлежащие проверке, находятся в аккумуляторе. Они будут утеряны.

4. OR I,Данные, где Данные = FF(1111 1111).

5. Команда OR B осуществляет упаковку двух слов и вырабатывает содержимое аккумулятора 1001 1000.

6. Вначале содержимое аккумулятора обрабатывается командой И совместно с маской 0000 1111:

1111 1000	Данные
0000 1111	Маска
0000 1000	Данные И Маска

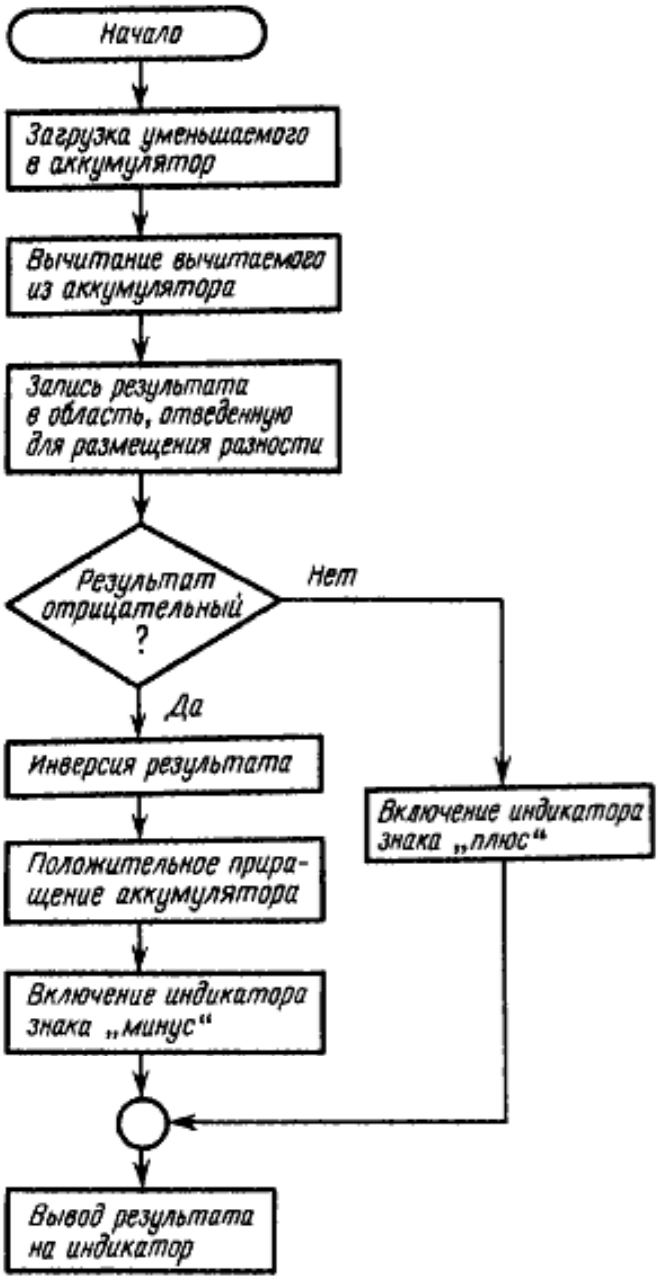


Рис. 9.15. Блок-схема к ответу на вопрос п. 8 заданий для самопроверки.

Затем маскированное слово было передано в регистр B с помощью команды MOV B,A.

7. Эта логическая связь определяется операцией И-НЕ.

8. См. рис. 9.15.

9. а) CMP I,03 заменяется на CMP I,04;

б) новый блок проверки условия показан на рис. 9.16, схема распределения памяти не изменяется;

в) необходимо ввести в программу еще одну команду CMP I, за которой следует еще одна команда JZ.

10. При выполнении команд ИСКЛЮЧАЮЩЕЕ ИЛИ и ВЫЧИТАНИЕ результат загружается в аккумулятор. При выполнении команды СРАВНЕНИЕ результат как таковой не формируется: производится лишь соответ-

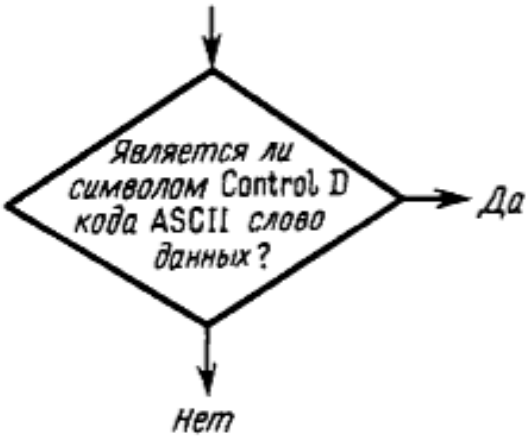
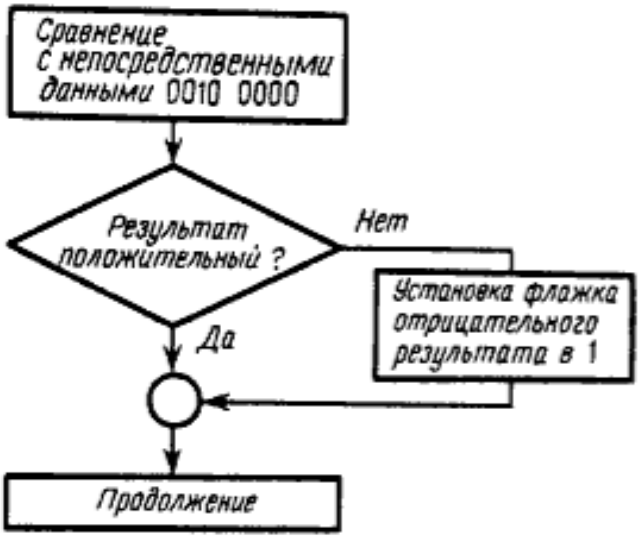


Рис. 9.16. Блок проверки условия к ответу на вопрос п. 9, б заданий для самопроверки.



ствующая установка разрядов регистра состояния.

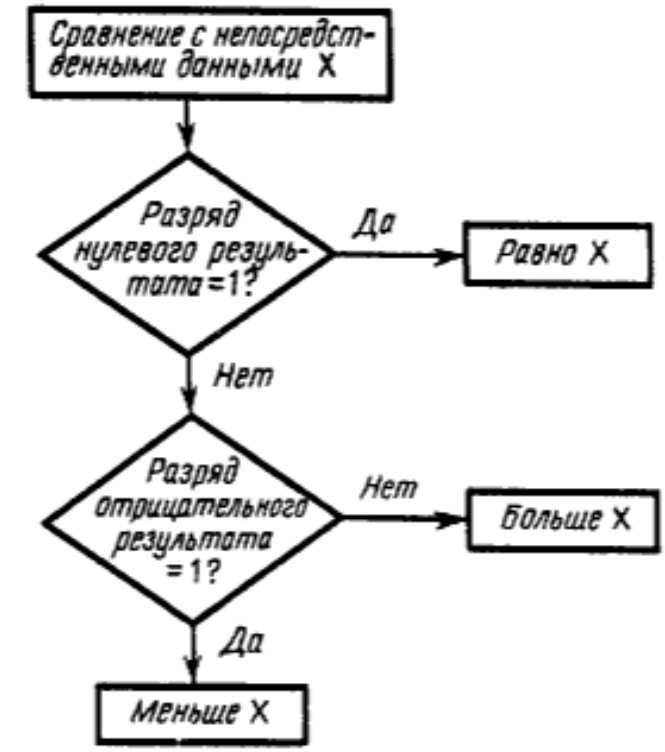


Рис. 9.18. Блок-схема к ответу на вопрос п. 12 заданий для самопроверки.

Рис. 9.17. Блок-схема к ответу на вопрос п. 11 заданий для самопроверки.

11. См. рис. 9.17.
12. См. рис. 9.18.

13.

	C	Аккумулятор
а	1	10101000
б	1	01010000
в	0	11101010
г	0	11110101

	C	Аккумулятор
в	1	10101000
е	1	01010001
ж	0	01101010
з	0	00110101

14.

	C	Аккумулятор
а	0	01011110
б	0	10111100
в	1	00010111
г	1	00001011

	C	Аккумулятор
д	0	01011110
е	0	10111100
ж	1	00010111
з	1	10001011

15.

а	0	1	1	1	1	1	1	1	0
б	1	1	1	1	1	1	1	0	0
в	1	0	0	1	1	1	1	1	1
г	1	0	0	0	1	1	1	1	1
д	0	1	1	1	1	1	1	1	1
е	1	1	1	1	1	1	1	1	0
ж	1	1	0	1	1	1	1	1	1
з	1	1	1	0	1	1	1	1	1

16.

а	1	1	1	1	1	1	1	0	0
б	1	1	1	1	1	1	0	0	0
в	0	1	1	1	1	1	1	1	1
г	1	1	1	1	1	1	1	1	1
д	1	1	1	1	1	1	1	0	0
е	1	1	1	1	1	1	0	0	1
ж	0	0	1	1	1	1	1	1	1
з	1	0	0	1	1	1	1	1	1

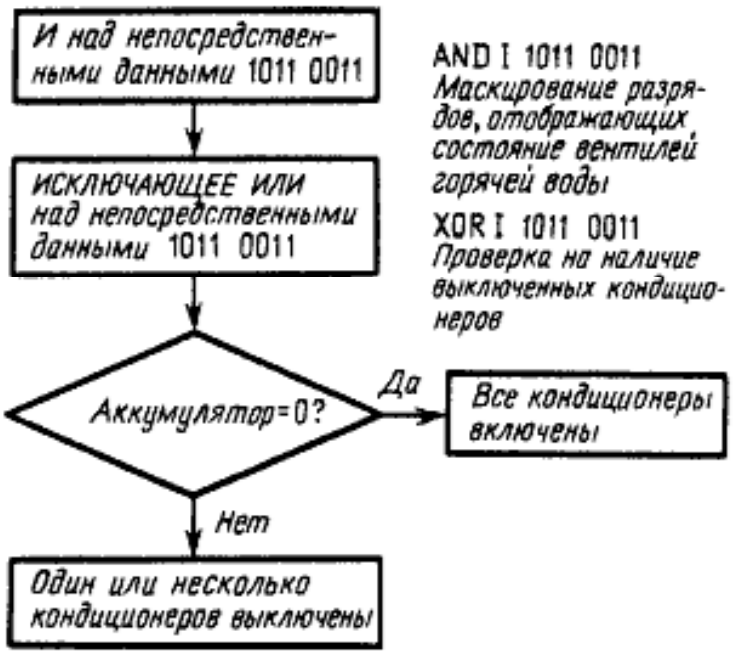


Рис. 9.19. Блок-схема подпрограммы к ответу на вопрос п. 17 заданий для самопроверки.

17. См. рис. 9.19. С помощью команды И маскируются, т. е. приводятся в состояние 0 все разряды, не служащие для обозначения состояния кондиционеров. По команде ИСКЛЮЧАЮЩЕЕ ИЛИ разряды, отведенные для указания состояния кондиционеров и имевшие значение 1 (что соответствует включенному состоянию этих устройств), устанавливаются в 0.

Глава 10. Команды перехода и вызова подпрограмм

Команды перехода и вызова подпрограмм позволяют изменять последовательность выполнения команд программы. Существуют два способа изменения этой последовательности. Первый из них называется *безусловным*. Согласно этому способу, последовательность выполнения программы подвергается изменению всякий раз, когда реализуется определенная команда. В соответствии со вторым способом последовательность выполнения программы определяется некоторыми *условиями*, т.е. изменяется в том и только в том случае, когда значение указанного условия совпадает с заданным.

Широкие возможности микропроцессора в значительной степени определяются его способностью принимать решения. Команды перехода и вызова подпрограмм являются одной из составных частей процесса принятия решений. В результате выполнения команд сравнения, арифметической и логической обработки данных вырабатываются значения разрядов нулевого и отрицательного результатов и разряда переноса в регистре состояния, причем зачастую оказываются установленными в 1 более чем один из этих разрядов. Команды перехода и вызова подпрограмм проверяют значения разрядов регистра состояния и определяют последующий ход выполнения программы в зависимости от результатов проверки.

Команды перехода, называемые также *командами ветвления*, позволяют организовать в программах циклы и разветвления. Команды вызова подпрограмм дают возможность сократить объем разрабатываемых программ за счет повторного использования подпрограмм. Наличие в микропроцессоре стека с указателем стека делает возможным возврат в главную программу после выполнения подпрограммы.

10.1. Команды перехода или ветвления

Команды перехода или ветвления расширяют возможности микропроцессора, позволяя принимать решения в ходе выполнения программы. В некоторых из рассмотренных выше примеров программ было показано, как используются такие команды. Они определяют последующее направление выполнения программы в зависимости от результата проверки значений условий. Проверке подлежат значения разрядов переноса, нулевого и отрицательного результатов.

Подобные команды нашего гипотетического микропроцессора мы будем называть командами перехода. Аналогичные команды, используемые в некоторых типах микропроцессоров, могут называться командами ветвления. Как правило, между этими классами команд различия незначительны или вообще отсутствуют. Однако, работая с конкретным микропроцессором, необходимо тщательно ознакомиться с материалами фирмы-изготовителя, относящимися к его набору команд, чтобы выяснить,

что имеется в виду в каждом случае под командой перехода или ветвления.

Разница между командами перехода сводится к различию используемых способов адресации и проверяемых в ходе выполнения этих команд признаков результата. В разных микропроцессорных семействах применяются самые разнообразные команды перехода.

Ниже будут рассмотрены используемые микропроцессором команды перехода. Каждая из них имеет длину 3 байт и выполняется за пять микроциклов. Во всех этих командах применяется прямая адресация. Команды перехода большинства реальных микропроцессоров также ограничиваются использованием прямой адресации.

В командах типа ПЕРЕХОД ПРЯМОЙ 2 байт, следующих за кодом операции перехода и представляющих собой адрес, указывают на некоторую область памяти. В указываемой области памяти содержится команда, которая должна выполняться следующей.

Эта область памяти может находиться как в непосредственной близости от текущей используемой области, так и значительном удалении от нее. Когда выполняется команда перехода, то не остается никаких сведений о том, в какой точке произошел выход из программы. Если впоследствии необходимо вернуться в исходную программу, следует каким-то образом сохранить сведения о точке выхода. При реализации команд другого типа запоминание адреса возврата происходит автоматически. Такие команды используются, когда нужно возвратиться в исходную программу после выполнения подпрограммы.

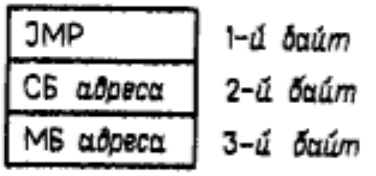
При использовании команды перехода изменяется содержимое счетчика команд. Содержимое второго и третьего байтов команды перехода пересылается в счетчик команд во время фазы выполнения. Тогда при очередном цикле выборки микропроцессор извлекает команду из области памяти, на которую указывают второй и третий байты команды перехода.

Таким образом, происходит переход в другую точку программы. Теперь выполняются одна за другой команды новой последовательности. Это продолжается до тех пор, пока не будет опять выполнена команда перехода.

Рассмотрим семь типичных команд перехода:

ПЕРЕХОД БЕЗУСЛОВНЫЙ

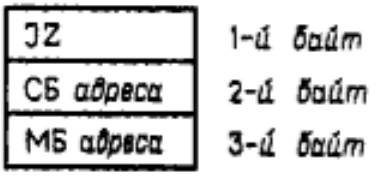
JMP,Адрес Старший байт адреса → → Старший байт СК
Младший байт адреса → → Младший байт СК



Название этой команды говорит о том, каким образом она выполняется. При этом не производится никаких проверок. В счетчик команд микропроцессора автоматически загружается содержимое второго и третьего байтов команды. Выполнение программы продолжается с этой новой исходной точки, когда начинается цикл выборки команды по указанному в команде адресу.

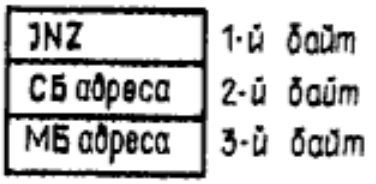
ПЕРЕХОД, ЕСЛИ НУЛЬ

JZ,Адрес
Если Z = 1, то Старший байт адреса → → Старший байт СК
Младший байт адреса → → Младший байт СК



ПЕРЕХОД, ЕСЛИ НЕ НУЛЬ

JNZ,Адрес
Если Z = 0, то Старший байт адреса → → Старший байт СК
Младший байт адреса → → Младший байт СК



Команды ПЕРЕХОД, ЕСЛИ НУЛЬ и ПЕРЕХОД, ЕСЛИ НЕ НУЛЬ проверяют

значение разряда нулевого результата регистра состояния. При выполнении команды ПЕРЕХОД, ЕСЛИ НУЛЬ содержимое счетчика команд изменяется в ситуации, когда значение указанного разряда регистра состояния имеет значение 1, т.е. когда равно 0 содержимое всех разрядов аккумулятора.

При выполнении команды ПЕРЕХОД, ЕСЛИ НЕ НУЛЬ содержимое счетчика команд изменяется, если разряд нулевого результата в регистре состояния имеет значение 0. Таким образом, в этом случае переход имеет место, когда не все разряды аккумулятора равны 0 (как известно, разряд нулевого результата регистра состояния имеет значение 0, если содержимое хотя бы одного из разрядов аккумулятора равно 1).

Иногда программисты называют команду ПЕРЕХОД, ЕСЛИ НУЛЬ командой ПЕРЕХОД, ЕСЛИ РАВНО (JE). Этим названием данная команда обязана тому обстоятельству, что ее часто применяют сразу же после команды СРАВНЕНИЕ. Если два сравниваемых числа оказываются равны, разряд нулевого результата в регистре состояния устанавливается в 1. В этом случае в программе имеет место переход. Ясно, почему команду ПЕРЕХОД, ЕСЛИ НУЛЬ можно рассматривать как команду ПЕРЕХОД, ЕСЛИ РАВНО. Аналогично команду JNZ можно представлять себе как команду ПЕРЕХОД, ЕСЛИ НЕ РАВНО (JNE).

ПЕРЕХОД, ЕСЛИ ПЕРЕНОС

JC,Адрес

Если C = 1, то Старший байт адреса → → Старший байт СК
Младший байт адреса → → Младший байт СК

JC	1-й байт
СБ адреса	2-й байт
МБ адреса	3-й байт

ПЕРЕХОД, ЕСЛИ НЕТ ПЕРЕНОСА

JNC,Адрес

Если C = 0, то Старший байт адреса → → Старший байт СК
Младший байт адреса → → Младший байт СК

JNC	1-й байт
СБ адреса	2-й байт
МБ адреса	3-й байт

При выполнении команд ПЕРЕХОД, ЕСЛИ ПЕРЕНОС и ПЕРЕХОД, ЕСЛИ НЕТ ПЕРЕНОСА проверяется значение разряда переноса регистра состояния. Когда выполняется команда ПЕРЕХОД, ЕСЛИ ПЕРЕНОС, то содержимое ее второго и третьего байтов загружается в счетчик команд, если разряд переноса регистра состояния установлен в 1.

В процессе реализации команды ПЕРЕХОД, ЕСЛИ НЕТ ПЕРЕНОСА содержимое ее второго и третьего байтов загружается в счетчик команд, если разряд переноса регистра состояния имеет значение, равное 0. Обе эти команды часто применяются сразу после выполнения арифметических команд.

ПЕРЕХОД, ЕСЛИ МИНУС

JN,Адрес

Если N = 1, то Старший байт адреса → → Старший байт СК
Младший байт адреса → → Младший байт СК

JN	1-й байт
СБ адреса	2-й байт
МБ адреса	3-й байт

ПЕРЕХОД, ЕСЛИ НЕ МИНУС

JNN,Адрес

Если N = 0, то Старший байт адреса → → Старший байт СК
Младший байт адреса → → Младший байт СК

JNN	1-й байт
СБ адреса	2-й байт
МБ адреса	3-й байт

Команда ПЕРЕХОД, ЕСЛИ МИНУС выясняет значение разряда отрицательного результата регистра состояния. Если он оказывается установленным в 1, то данная команда осуществляет переход в программе. Управление программой передается по новому адресу.

Команда ПЕРЕХОД, ЕСЛИ НЕ МИНУС также проверяет значение разряда отрицательного результата; она реализует переход, если разряд находится в состоянии 0. Эту команду можно назвать также ПЕРЕХОД, ЕСЛИ ПЛЮС. Очевидно, если разряд отрицательного результата имеет значение 0, то содержимое аккумулятора представляет собой положительное число.

Задания для самопроверки

1. Изобразите в виде блок-схем алгоритмов процессы принятия решений «больше чем» и «равно». Какие команды понадобятся для этой цели?

2. Регистр B микропроцессора используется в качестве счетчика, содержимое которого уменьшается. Как произвести начальную установку этого счетчика? Каким образом определить, что его содержимое уменьшилось на 10? Изобразите блок-схему алгоритма и укажите используемые команды.

3. В какой области (а) находится команда, которой передает управление команда JNN 006A, находящаяся в области памяти 012A, если командой, находящейся в области 0129, был произведен сброс одного из регистров (б) микропроцессора? Объясните.

4. В какой области (а) находится команда, которой передается управление при условиях, указанных в п. 3, если командой, находящейся в области 0129, старший разряд того же регистра (б) был установлен в 1? Объясните.

5. а) Какое другое название может быть использовано для команды JE? б) За какой командой данная команда часто следует в программе?

6. В области памяти 00AC находится команда СЛОЖЕНИЕ С ПАМЯТЬЮ КОСВЕННОЕ. По этой команде число 01 прибавляется к числу FF, расположенному в аккумуляторе. За ней следует команда JC 00B9. Что происходит при выполнении команды, находящейся в области 00AD? Изобразите блок-схему программы.

10.2. Пример программы

Продемонстрируем некоторые способы применения команд перехода на примере. Рассмотрим, как используются команды перехода для сортировки данных, изменения последовательности выполнения программы и выяснения состояния счетчиков, содержимое которых подвергается уменьшению. Пример иллюстрирует, кроме того, некоторые другие приемы составления программ: использование регистра в качестве указателя памяти, работу с буферными регистрами и многократное использование одной и той же команды.

Схема распределения памяти для рассматриваемой программы приведена на рис. 10.1, а ее блок-схема – на рис. 10.2.

Выясним вначале, какие действия выполняет программа. Она начинается с области памяти с адресом 0000 и заканчивается в области с адресом 0025, содержащей ко-

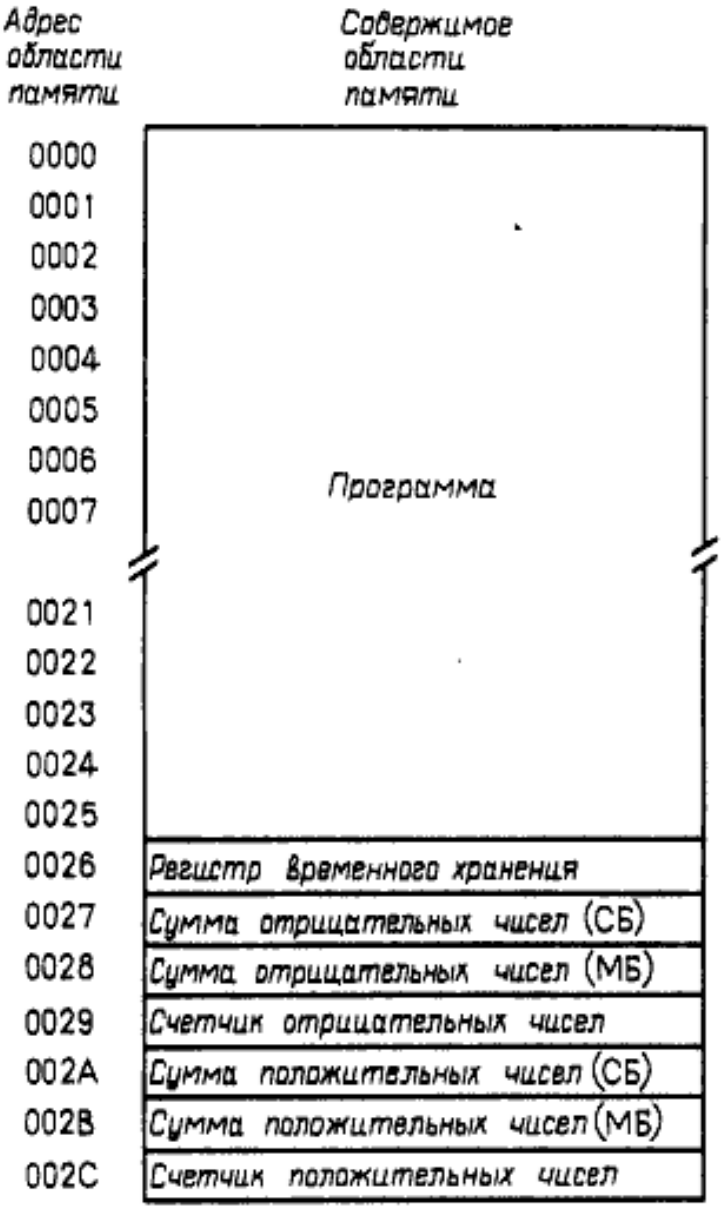


Рис. 10.1. Схема распределения памяти для программы сортировки данных. (Под данные отведены области памяти с 0026 по 002C.)

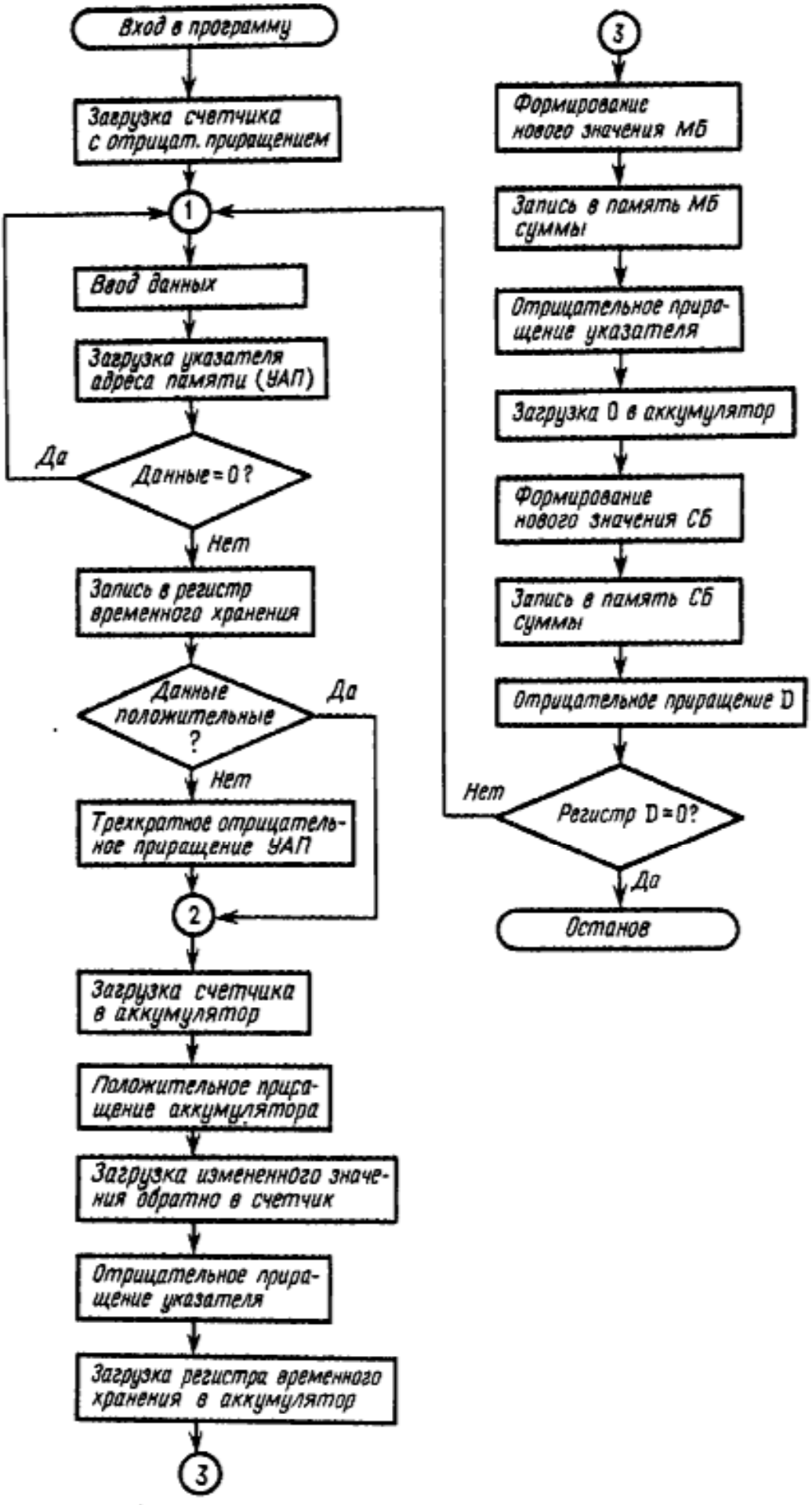


Рис. 10.2. Блок-схема программы сортировки данных.

манду ОСТАНОВ. Весьма вероятно, что впоследствии вы будете ее использовать как стандартную подпрограмму. Такой подпрограмме передается управление со стороны более сложной программы, когда необходимо реализовать определенную функцию.

Блок-схема программы дает представление о том, как можно решить данную конкретную задачу с помощью микропроцессора. Однако, прежде чем строить блок-схему, необходимо разработать алгоритм решения задачи. *Алгоритм* – это совокупность правил или процессов, обеспечивающих решение задачи за определенное число шагов. Блок-схема является способом графического представления алгоритма. Так, рис. 10.2 является графическим изображением рассматриваемого здесь варианта алгоритма решения данной задачи.

Назначением программы является сортировка данных, представленных в дополнительном коде, по трем категориям: положительные числа, отрицательные числа и числа, равные 0. Если данные имеют нулевое значение, они не используются. Положительные и отрицательные числа прибавляются к соответствующим суммам, хранимым в памяти. Впоследствии полученные таким образом суммы используются для определения среднего значения положительных и отрицательных чисел, являющихся результатом проведенных измерений.

В процессе выполнения программы используются два счетчика. Один из них служит для подсчета количества положительных чисел, а второй – для подсчета количества отрицательных чисел.

Имеется еще один счетчик (регистр D) с уменьшением содержимого при подсчете, в который заносится в начале программы число FF. Содержимое счетчика подвергается отрицательному приращению каждый раз, когда обнаруживается допустимое (положительное или отрицательное) значение измеряемых данных. Установка регистра D в 0 означает, что было выполнено 256 измерений. На этом работа программы заканчивается.

Предположим, что данные представляют собой 8-битовые величины со знаком, поступающие от аналого-цифрового преобразователя (АЦП). С выхода АЦП поступают числа в дополнительном коде, старший бит которых является знаковым. Остальные 7 бит – это двоичное число, представляющее уровень аналогового сигнала. Для данного АЦП код 0111 1111 соответствует напряжению + 1,27 В, код 0000 0000 – напряжению 0 В, а код 1000 0000 – напряжению – 1,28 В.

При выполнении команды ВВОД в аккумулятор микропроцессора передается из АЦП одно 8-битовое выходное слово.

Схема распределения памяти, приведенная на рис. 10.1, отражает использование памяти микро-ЭВМ при работе с рассматриваемой программой.

Область памяти с адресом 0026 играет роль регистра временного хранения. В нее необходимо записывать данные на то время, пока микропроцессор занят выполнением других операций. Указанные регистры обычно служат, как и в данном случае, для предотвращения потери данных во время реализации промежуточных шагов программы.

В области памяти с адресом 002C размещается счетчик положительных чисел. В начале программы содержимое этой области равно 0. Каждый раз, когда измеренные данные оказываются положительной величиной, значение счетчика увеличивается на 1.

Область памяти 002B служит для хранения младшего байта суммы положительных результатов измерений. Каждый раз, когда измеренные данные оказываются положительными, содержимое области 002B складывается со словом данных. Результат этого сложения, т. е. обновленная сумма положительных чисел, записывается опять в область 002B. Старое значение суммы при этом, конечно, теряется.

Количество положительных результатов измерений может достигать 256, причем каждый из них может быть представлен с помощью 7 бит. Поэтому для размещения суммы необходимо отвести более 8 бит. Цифры суммы, выходящие за пределы области 002B, записываются в область памяти 002A. После выполнения каждого измерения и записи в память значения младшего байта суммы значение ее старшего байта, находящегося в области 002A, складывается с переносом, имевшим место при сложении младшего байта суммы и данными измерений.

Указанные 3 байт данных и бит переполнения составляют полную информацию о положительных результатах измерений. Данные, содержащиеся в областях памяти 0029, 0028 и 0027, составляют аналогичную информацию об отрицательных результатах измерений.

Как следует из схемы распределения памяти, для размещения всей 38-байтовой программы используются области памяти с 0000 по 0025.

Блок-схема алгоритма, представленная на рис. 10.2, поясняет последовательность выполнения программы. Полный листинг программы приведен на рис. 10.3. Оба этих рисунка необходимо использовать в ходе ознакомления с программой. Листинг программы хорошо «документирован». По сути дела содержащихся в нем сведений может оказаться достаточно для понимания программы. Однако программисты редко документируют свои программы столь же тщательно. Читателю следует использовать предоставляемую здесь возможность, чтобы научиться «читать между строк». Часто приходится это делать при ознакомлении с другими листингами программ.

Программа сортировки довольно широко

Адрес области памяти	Содержимое области памяти	Комментарий
0000	LDA D	Загрузка 256_{10} в основной счетчик (регистр D)
0001	FF	256_{10}
0002	IN	ВВОД
0003	01	Порт 01
0004	JZ	Возврат к команде ВВОД, если данные равны 0
0005 0006	00 02	
0007	LRP B	Установка указателя (пары BC) на начало файла
0008	00	
0009	2C	
000A	STA A	Перепись данных в регистр временного хранения
000B	00	
000C	26	
000D	JNN	Данные положительные? Да, пропуск отрицательных приращений
000E	00	
000F	13	

Рис. 10.3. Листинг программы сортировки данных. (Начальные значения данных в отведенной для них области памяти установлены равными 0.)

Адрес области памяти	Команда	Комментарий
0010	DRP B	Данные отрицательные?
0011	DRP B	
0012	DRP B	
0013	LDI A	Загрузка счетчика в аккумулятор
0014	INC A	Прибавление 1
0015	STI A	Запись нового значения счетчика в память
0016	DRP B	Установка указателя на МБ
0017	LDD A	Извлечение данных из регистра временного хранения
0018	00	
0019	26	
001A	ADI M	Формирование нового значения МБ суммы
001B	STI A	Запись в память нового значения МБ суммы
001C	DRP B	Установка указателя на СБ
001D	LDA A	Загрузка 0 (СБ данных)
001E	00	
001F	ACI M	Формирование нового значения СБ суммы с учетом переноса
0020	STI A	Запись в память нового значения СБ суммы
0021	DEC D	Отрицательное приращение основного счетчика
0022	JNZ	Равно 0? Нет, возврат к команде ВВОД
0023	00	
0024	02	
0025	HLT	Равно 0? Да, выход из программы
0026	00	Регистр временного хранения
0027	00	СБ суммы отрицательных результатов измерений

Рис. 10.3. Продолжение.

Адрес области памяти	Содержимое области памяти	Комментарий
0028	00	МБ суммы отрицательных результатов измерений
0029	00	Счетчик количества отрицательных чисел
002A	00	СБ суммы положительных результатов измерений
002B	00	МБ суммы отрицательных результатов измерений
002C	00	Счетчик количества положительных чисел

Рис. 10.3. Продолжение.

применяется. В ней используются две команды условного перехода. Первая из них проверяет, не равно ли 0 значение данных. Если данные не равны 0, то они либо положительные, либо отрицательные. Чтобы выяснить это, выполняется их очередная проверка. Она устанавливает, является ли значение данных положительным. Очевидно, если оно не положительное, значит, данные отрицательные. Таким образом, с помощью двух команд условного перехода осуществляется сортировка данных по трем типам.

В начале программы по команде ЗАГРУЗКА НЕПОСРЕДСТВЕННАЯ в регистр D загружается исходное значение FF. Этот регистр используется в качестве основного счетчика. Его содержимое подвергается отрицательному приращению после каждого учитываемого измерения. Когда регистр D устанавливается в 0, то выполнение программы прекращается. В случае необходимости можно в этом месте программы поместить последовательность команд, присваивающих значение, равное 0, содержимому областей с адресами 0026–002C.

При реализации команды ВВОД слово данных пересылается из порта ввода-вывода 01 в аккумулятор.

Теперь необходимо загрузить исходное значение в регистровую пару BC. Это делается каждый раз после выполнения команды ВВОД. Регистровая пара BC указывает на области памяти, в которых размещаются суммы положительных и отрицательных чисел, а также счетчики количества выполненных измерений. Вначале в регистры BC заносится значение, являющееся адресом счетчика положительных чисел. Этот счетчик размещается в области памяти 002C. Трехкратно выполняемая команда ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ РЕГИСТРОВОЙ ПАРЫ служит для того, чтобы последовательно указывать на области памяти, в которых размещаются сумма результатов и счетчик отрицательных чисел.

Первая проверка выполняется с помощью команды JZ. Если значение данных равно 0, то управление передается опять команде ВВОД. Это означает, что будет осуществлен ввод новых данных.

Если значение данных не равно 0, выполнение программы продолжается по другому направлению. По команде ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ ПРЯМАЯ копия данных записывается в область памяти с адресом 0026. Как будет ясно из дальнейшего изложения, делается это потому, что данные, находящиеся в аккумуляторе, будут уничтожены. Когда эти данные потребуются, можно будет взять их копию из регистра временного хранения.

Следующее решение принимается по результатам выяснения факта, являются ли данные положительными. Если при выполнении команды JNN (ПЕРЕХОД, ЕСЛИ НЕ МИНУС) выясняется, что данные являются положительной величиной, то три команды отрицательного приращения пропускаются и происходит переход к подпрограмме суммирования положительных чисел. Если же

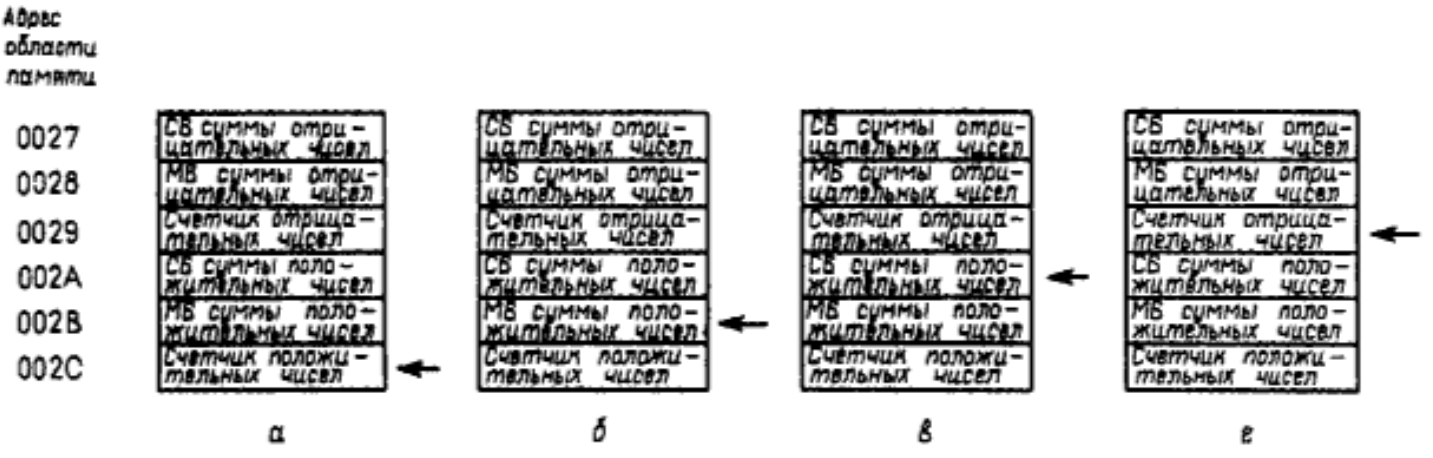


Рис. 10.4. Отрицательные приращения содержимого регистровой пары BC: *а* – исходное содержимое, имеющее место после выполнения команды LRP B,002C; *б* – содержимое регистров после выполнения первой команды DRP; *в* – содержимое регистров после выполнения второй команды DRP; *г* – содержимое регистров после выполнения третьей команды DRP.

данные оказываются величиной отрицательной, то выполнение программы продолжается в естественном порядке, так как переход отсутствует. В этом случае реализуются три команды отрицательного приращения, после чего программа выполняется в той же последовательности, как и при положительных данных.

Если данные представляют собой отрицательную величину, то содержимое пары регистров BC трижды подвергается отрицательному приращению. На рис. 10.4 показано, что в результате этого действия пара регистров BC указывает на счетчик отрицательных чисел. В этом случае можно сразу начинать выполнение подпрограммы суммирования.

Так как пара регистров BC указывает на счетчик отрицательных чисел, можно прибегнуть к команде ЗАГРУЗКА АККУМУЛЯТОРА КОСВЕННАЯ. Содержимое счетчика отрицательных чисел переписывается при выполнении этой команды в аккумулятор. С помощью команд INC A и STI A к значению счетчика прибавляется двоичная 1, и образованное таким образом новое значение загружается обратно в область с адресом 0029.

Затем содержимое регистров BC еще раз подвергается отрицательному приращению. Такое применение регистровой пары BC позволяет использовать последовательность команд с косвенной адресацией для пересылки данных в общую область памяти и из нее. В этой области (0028) содержится младший байт суммы отрицательных чисел.

Теперь нам потребуются данные, которые находились ранее в аккумуляторе. На место этих данных в аккумулятор было записано значение счетчика отрицательных чисел, но мы предварительно переписали их в регистр временного хранения. По команде LDA A исходные данные вновь помещаются в аккумулятор. Теперь можно прибавить их к сумме отрицательных чисел.

По команде СЛОЖЕНИЕ С ПАМЯТЬЮ КОСВЕННОЕ к данным, находящимся в аккумуляторе, прибавляется содержимое области памяти 0028. Если необходимо, то устанавливается в 1 разряд переноса регистра состояния. Затем полученная сумма записывается вновь в область с адресом 0028 с помощью команды ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ КОСВЕННАЯ.

Как мы уже выяснили, результат такого сложения может иметь длину до 16 бит. Поэтому нужно теперь сформировать старший байт суммы. После еще одного отрицательного приращения регистры BC указывают на местоположение в памяти старшего байта суммы отрицательных чисел. Мы знаем, что выходные данные АЦП не содержат второго (старшего) байта. Другими словами, их старший байт содержит 0 во всех разрядах. Поэтому в аккумулятор загружается число 00. Делается это с помощью команды LDA A, в выполнении которой не участвует бит разряда переноса регистра состояния.

Теперь выполняются команда СЛОЖЕНИЕ С ПАМЯТЬЮ И ПЕРЕНОСОМ КОСВЕННОЕ, т. е. производится сложение ранее сформированного старшего байта

суммы, бита переноса и старшего байта выходных данных АЦП (число 00). Результат записывается в область памяти 0027 с помощью команды ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ КОСВЕННАЯ.

На этом заканчивается выполнение подпрограммы суммирования отрицательных чисел. После нее выполняется простая подпрограмма отрицательного приращения содержимого регистра D. После отрицательного приращения содержимого регистра D команда JNZ проверяет, равен ли 0 разряд нулевого результата в регистре состояния. Если значение этого разряда равно 0, то происходит переход к команде ВВОД, находящейся в области памяти с адресом 0002. Если команда JNZ обнаруживает, что значение разряда нулевого результата не равно 0, то переход не осуществляется. В этом случае в качестве очередной выполняется команда ОСТАНОВ (HLT).

На этом выполнение программы завершается. Возможно, читателю целесообразно еще раз проследить последовательность действий, реализуемых каждой командой. В рассмотренной программе использованы некоторые приемы, которые широко применяются при работе с микропроцессорами.

В ходе более глубокого ознакомления с программой могут быть найдены другие, даже более рациональные способы решения той же задачи. Эта программа так же, как и ряд программ, рассматриваемых в книге, призвана ознакомить читателя с определенными концепциями программирования. Она не являет собой пример наиболее компактной программы. Необходимо всегда помнить, что для решения любой задачи имеется всегда более чем один способ.

Задания для самопроверки

7. Укажите последовательность изменения содержимого счетчика команд при выполнении программы, блок-схема и листинг которой приведены соответственно на рис. 10.2 и 10.3, при обработке отрицательных, положительных и нулевых данных. Приведите эту последовательность для одного цикла.

8. Разработайте блок-схему программы, предназначенной для записи данных в один из трех файлов. Файл, начинающийся с области 0100, должен включать числа, значения которых меньше или равны 0000 1000. Файл, который размещается в памяти, начиная с области 0200, должен содержать числа, значения которых лежат в диапазоне от 0000 1001 до 0010 0000. Файл, начинающийся с области 0300, должен охватывать числа, значения которых больше или равны 0010 0001. Укажите команды микропроцессора, которые следует применить для реализации шагов составленного алгоритма. Программа должна осуществлять учет следующих величин:

а) количества чисел, значения которых не превышают 0000 1000 (для этой величины отводится область памяти 00FB);

б) количества чисел, значения которых лежат в диапазоне от 0000 1001 до 0010 0000 (данная величина содержится в области 00FC);

в) количества чисел, значения которых больше или равны 0010 0001 (эта величина должна размещаться в области 00FD).

9. Доработайте блок-схему, построенную при ответе на вопрос п. 8 заданий, таким образом, чтобы программа осуществляла подсчет общего числа измеренных значений. Это число должно размещаться в области памяти 00FE. Кроме того, выполнение программы должно прекращаться, когда общее число измерений превышает 100.

10.3. Команды вызова подпрограмм

Мы рассмотрели команды перехода. С их помощью последовательность выполнения программы может быть изменена путем перехода к новой последовательности. Как мы видели, характерным способом использования команд перехода является организация программных циклов. Однако команда перехода сама по себе не позволяет вернуться в то место главной программы, откуда был осуществлен переход.

Существует множество ситуаций, когда необходимо осуществить выход из главной программы, но впоследствии опять вернуться в нее. При этом возврат необходимо произвести именно в то место, из которого произошел переход. Один из способов решения

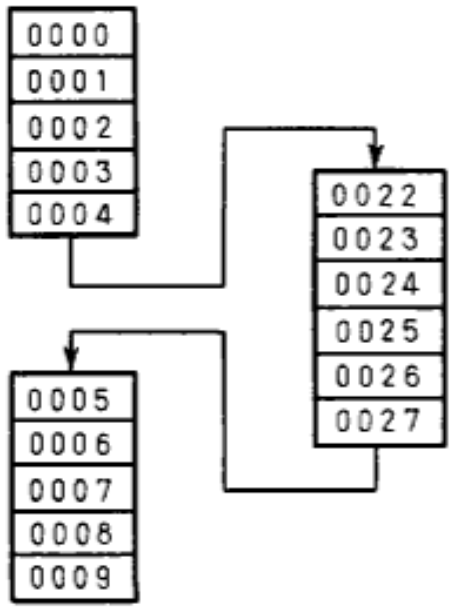


Рис. 10.5. Использование подпрограммы с последующим возвратом в главную программу.

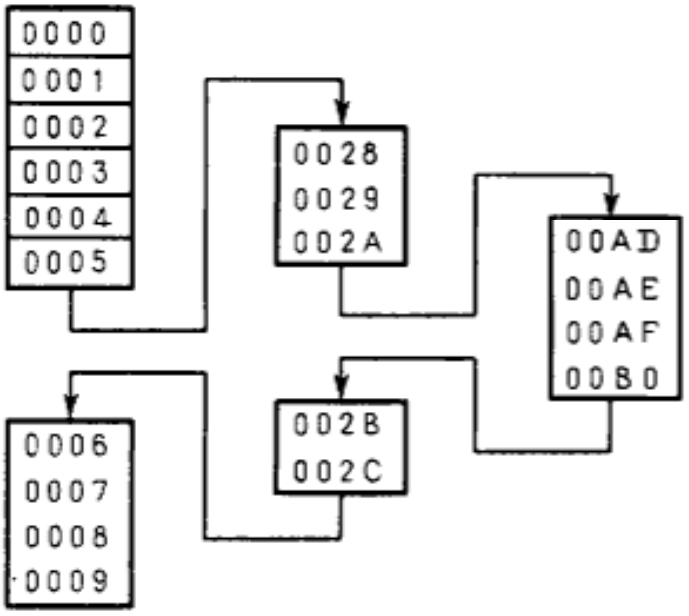


Рис. 10.6. Организация вложения подпрограмм.

этой проблемы заключается в использовании второй команды перехода. С помощью команды безусловного или условного перехода происходит переход к подпрограмме, а по завершении выполнения подпрограммы посредством другой команды перехода осуществляется возврат в ту точку главной программы, из которой был сделан переход. Однако для этого необходимо знать заранее, в какой точке главной программы произошел переход.

Легче добиться той же цели с помощью команды вызова подпрограммы. Подпрограмма представляет собой фрагмент программы, обращение к которой может иметь место в любой точке главной программы. Когда происходит вызов подпрограммы, то в начале своего выполнения она реализует действия по запоминанию текущего содержимого счетчика команд. Когда выполнение подпрограммы заканчивается, то с помощью всего лишь одной команды микропроцессору указывается, что исходное содержимое счетчика команд должно быть извлечено из памяти. Этой информации микропроцессору достаточно, чтобы осуществить возврат в прерванную последовательность команд главной программы.

Рис. 10.5 иллюстрирует действия подпрограммы. Первые пять шагов программы (команды, находящиеся в областях с адресами с 0000 по 0004) выполняются друг за другом. Команда, хранимая в области 0004, представляет собой команду ВЫЗОВ ПОДПРОГРАММЫ. При ее выполнении счетчик команд указывает на команду, находящуюся в области памяти с адресом 0022. Теперь последовательно друг за другом реализуются команды, находящиеся в областях 0022–0027. В области 0027 находится команда ВОЗВРАТ ИЗ ПОДПРОГРАММЫ, в соответствии с которой счетчик команд обращается вновь к тому месту программы, из которого произошло обращение к подпрограмме, и указывает теперь на область 0005. Выполнение программы продолжается далее в виде реализации последовательности команд, хранимых в областях с 0005 по 0009.

На рис. 10.6 показан пример более сложных действий с подпрограммами. Вначале происходит последовательное выполнение первых шести команд главной программы. Затем при реализации команды в области с адресом 0005 микропроцессор ставится в известность, что следующей очередной командой должна являться команда, находящаяся в области 0028. Таким образом, шестая команда производит вызов подпрограммы, которая хранится в памяти, начиная с области с адресом 0028. Далее последовательно выполняются команды, находящиеся в областях памяти 0028, 0029 и 002A. Затем в качестве адреса очередной команды в счетчик команд заносится номер области 00AD.

Тем самым опять осуществляется обращение к подпрограмме. Эта вторая подпрограмма выполняется в ходе реализации первой. Последовательно выполняются команды, находящиеся в областях памяти с адресами 00AD, 00AE, 00AF и 00B0. Затем в счетчик команд загружается адрес возврата. Делается это с помощью команды ВОЗВРАТ ИЗ ПОДПРОГРАММЫ, которая хранится в области 00B0. В соответствии с данной командой управление передается команде, находящейся в области памяти 002B. По команде ВОЗВРАТ ИЗ ПОДПРОГРАММЫ в счетчик команд заносится адрес команды, которая следует за последней реализованной командой ВЫЗОВ ПОДПРОГРАММЫ, несмотря на то что эта реализация имела место в процессе выполнения предшествующей подпрограммы.

Команды, расположенные в областях памяти с адресами 002B и 002C, выполняются одна за другой, затем в счетчик команд опять заносится адрес возврата, в результате чего он указывает на область памяти 0006. После этого последовательно выполняются команды, находящиеся в областях с адресами 0006–0009.

Описанный процесс называется *вложением* подпрограмм. Оно заключается в том, что главная программа обращается к некоторой подпрограмме, которая в свою очередь обращается к другой подпрограмме. Большинство микропроцессоров располагает возможностью практически неограниченного числа уровней вложения подпрограмм.

Рис. 10.7 иллюстрирует еще один способ использования подпрограмм. Здесь в процессе выполнения программы происходит троекратное обращение к одной и той же подпрограмме. Вначале последовательно выполняются команды, находящиеся в областях с 0000 по 0004. По команде, находящейся в области 0004, происходит вызов подпрограммы, начинающейся с области с адресом 00AD. Теперь последовательно выполняются команды, находящиеся в областях с 00AD по 00B2. По команде ВОЗВРАТ ИЗ ПОДПРОГРАММЫ счетчик команд реализует возврат в главную программу. После этого последовательно выполняются команды, хранимые в областях 0005–0010. Команда ВЫЗОВ ПОДПРОГРАММЫ, находящаяся в области 0010, реализуется обращением к ранее использованной подпрограмме. Опять последовательно выполняются команды, расположенные в областях с 00AD по 00B2. В соответствии с командой возврата счетчик команд опять осуществляет возврат в то место главной программы, из которого происходило обращение к подпрограмме. На этот раз реализация главной программы возобновляется с команды, находящейся в области 0011, так как именно на этот адрес указывал счетчик команд, когда микропроцессор выполнял команду, расположенную в области 0010.

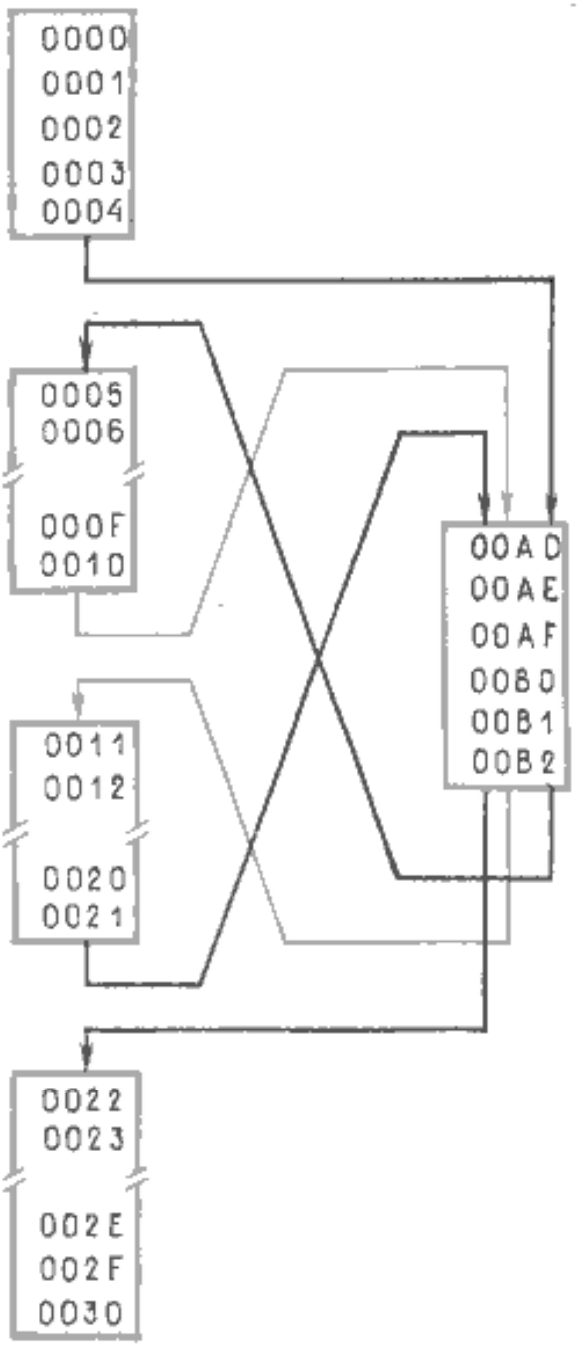


Рис. 10.7. Многократное использование одной подпрограммы.

Теперь последовательно выполняются команды, находящиеся в областях с 0011 по 0021, после чего опять имеет место вызов подпрограммы: по команде, хранимой в области 0021, происходит переход к команде, расположенной в области 00AD. Команды подпрограммы, которая размещена в областях с 00AD по 00B2, выполняются в последний, третий раз. После этого управление передается команде, находящейся в области памяти 0022. Программу завершает последовательность команд, размещенных в областях с 0022 по 0030.

В программах, которые проиллюстрированы рис. 10.5–10.7, использованы общие принципы применения подпрограмм. Организация подпрограмм позволяет обойтись без повторного написания одного и того же фрагмента программы. Конечно, таким путем достигается и экономия памяти. Общая повторяющаяся часть программы записывается в память также лишь один раз.

Предположим, что разработана короткая программа умножения двух чисел тройного формата. Такая программа может иметь длину от 30 до 50 байт. Вместо того чтобы переписывать ее заново столько раз, сколько она используется, достаточно написать ее лишь один раз. С помощью команд вызова подпрограмм можно использовать ее сколько угодно раз.

Во многих отношениях команды вызова подпрограмм аналогичны командам перехода. При выполнении команд вызова подпрограмм по условию выясняются значения разрядов регистра состояния; в зависимости от этих значений происходит или нет вызов подпрограмм.

При работе с подпрограммами используется стек микропроцессора. *Стек* – это область памяти, в которой временно сохраняется информация, необходимая для осуществления возврата в программу после выполнения подпрограммы. В некоторых микропроцессорах стек представляет собой набор внутренних регистров, т.е. регистров, размещенных в самом микропроцессоре. Однако стеки большинства микропроцессоров, в том числе рассматриваемого здесь, размещаются в памяти. Состояние стека отображается в каком-либо регистре, аналогичном регистровой паре BC и называемом указателем стека. *Указатель стека* содержит информацию о том, какая область памяти должна быть использована при очередном обращении к стеку. В большинстве микропроцессоров указатель стека оформлен в виде специального регистра.

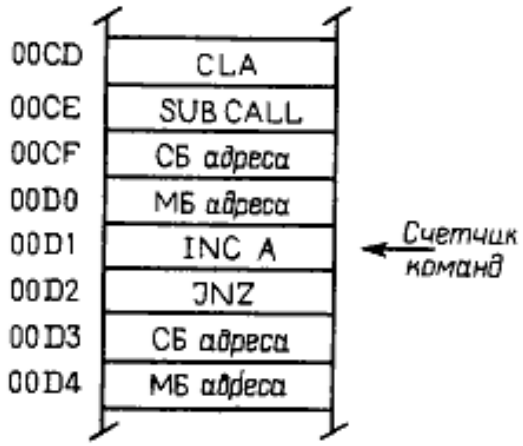


Рис. 10.8. Состояние счетчика команд при выполнении команды ВЫЗОВ ПОДПРОГРАММЫ. (Счетчик указывает на область 00D1, поскольку в ней находится команда, следующая за 3-байтовой командой ВЫЗОВ ПОДПРОГРАММЫ.)

При обращении к подпрограмме текущее содержимое счетчика команд загружается в стек. Как нам известно, текущее содержимое указывает местоположение следующей команды (рис. 10.8).

Каждый раз, когда в стек помещается 1 байт информации, осуществляется отрицательное приращение указателя стека. Стек заполняется в направлении убывания значений адресов памяти. Итак, при вызове подпрограммы текущее содержимое счетчика команд запоминается в стеке. Затем в счетчик команд загружается адрес области памяти, с которой должно начаться выполнение подпрограммы. Этот адрес берется из второго и третьего байтов команды вызова подпрограммы.

В то же время в стеке бережно сохраняется старое содержимое счетчика команд. Когда выполнение подпрограммы заканчивается, по команде ВОЗВРАТ значение указателя стека увеличивается на 1. Из стека извлекается хранившееся там содержимое счетчика команд.

Запись информации в стек микропроцессора называется *загрузкой* данных в стек, а чтение ее из стека – *извлечением* данных из стека.

Что происходит, когда в процессе выполнения одной подпрограммы происхо-

дит обращение из нее к другой подпрограмме? Такая ситуация была проиллюстрирована выше на рис. 10.6. При выполнении второй команды вызова подпрограммы текущее содержимое счетчика команд помещается в стек. Текущее содержимое счетчика команд представляет собой в данном случае адрес одной из команд первой подпрограммы. Эти 2 байт размещаются в двух областях стека, следующих за теми областями, в которых хранится текущее содержимое счетчика команд главной программы. Теперь в счетчик команд загружается из второго и третьего байтов команды вызова подпрограммы начальный адрес второй подпрограммы, и начинается выполнение подпрограммы второго уровня вложения. Состояние стека в этой ситуации иллюстрирует рис. 10.9.

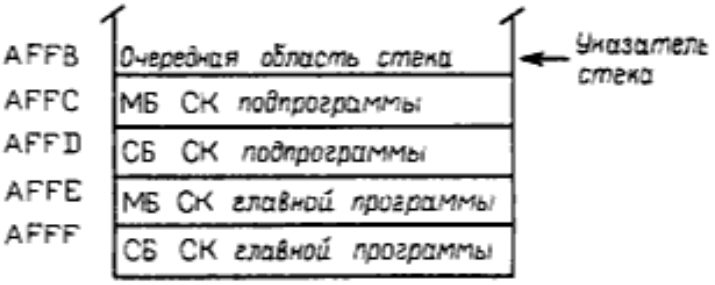


Рис. 10.9. Содержимое стека микропроцессора при двухуровневом вложении подпрограмм. (Стек начинается с адреса AFFF. Если в стек потребуется поместить еще какие-то данные, то они будут размещены по адресу AFFB, на что показывает указатель стека.)

Очевидно, что возможный уровень вложения подпрограмм определяется числом областей–ячеек, которыми располагает стек. Следует помнить, что пространство памяти, отводимое для стека, не может одновременно использоваться другими частями программы.

Указатель стека 8-разрядного микропроцессора обычно имеет длину 16 бит. Следовательно, в случае необходимости стек может занимать до 65 536 байт памяти. Однако на практике, если объем стека чрезмерно возрастает, то происходит его наложение на рабочую область памяти. При искажении информации, находящейся в стеке, работа микропроцессорной системы нарушается, так как микропроцессор не может вернуться к выполнению исходной программы.

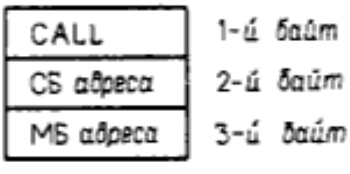
Когда выполнение подпрограммы заканчивается, из стека извлекается значение счетчика команд. По завершении подпрограммы второго уровня вложения из стека изымается значение счетчика команд подпрограммы первого уровня вложения. Оно загружается в счетчик команд, после чего выполнение указанной подпрограммы может быть возобновлено, начиная с команды, которая следовала за командой ВЫЗОВ ПОДПРОГРАММЫ. Когда завершается выполнение подпрограммы первого уровня, из стека опять извлекается очередное значение счетчика команд. Теперь возобновляется, начиная с команды, следовавшей за командой ВЫЗОВ ПОДПРОГРАММЫ в главной программе, выполнение главной программы.

Возврат к исходной точке программы или подпрограммы осуществляется с помощью одной команды, которая находится в конце вложенной подпрограммы. Эта команда называется ВОЗВРАТ ИЗ ПОДПРОГРАММЫ и описана ниже. Это 1-байтовая четырехцикловая команда. Рассматриваемый здесь микропроцессор снабжен лишь простой командой ВОЗВРАТ ИЗ ПОДПРОГРАММЫ, однако нередко микропроцессоры располагают различными командами возврата по выполнении некоторого условия. Команды ВЫЗОВ ПОДПРОГРАММЫ нашего гипотетического микропроцессора имеют длину 3 байт и реализуются за пять микроциклов. Во всех этих командах используется прямая адресация, тогда как в других, более сложных типах микропроцессоров в командах вызова подпрограмм применяются другие виды адресации.

Существуют следующие команды вызова подпрограмм:

ВЫЗОВ ПОДПРОГРАММЫ БЕЗУСЛОВНЫЙ

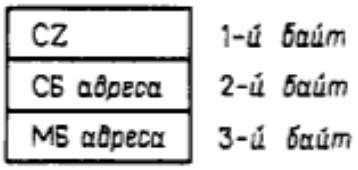
CAL,Адрес СК → Стек
Старший байт адреса → Старший байт счетчика команд
Младший байт адреса → Младший байт счетчика команд



При выполнении этой команды никаких проверок не производится, т.е. вызов подпрограммы происходит в любом случае.

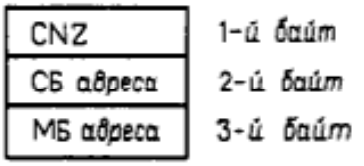
ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ НУЛЬ

CZ,Адрес Если Z = 1, то СК → Стек
Старший байт адреса → Старший байт счетчика команд
Младший байт адреса → Младший байт счетчика команд



ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ НЕ НУЛЬ

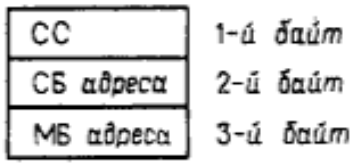
CNZ,Адрес Если Z = 0, то СК → Стек
Старший байт адреса → Старший байт счетчика команд
Младший байт адреса → Младший байт счетчика команд



При выполнении команд **ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ НУЛЬ** и **ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ НЕ НУЛЬ** проверяется значение разряда нулевого результата регистра состояния. По аналогии с соответствующими командами перехода данные команды можно называть **ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ РАВНО** и **ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ НЕ РАВНО.**

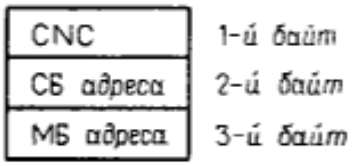
ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ ПЕРЕНОС

CC,Адрес Если C = 1, то СК → Стек
Старший байт адреса → Старший байт счетчика команд
Младший байт адреса → Младший байт счетчика команд



ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ НЕТ ПЕРЕНОСА

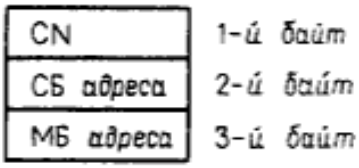
CNC,Адрес Если C = 0, то СК → Стек
Старший байт адреса → Старший байт счетчика команд
Младший байт адреса → Младший байт счетчика команд



При выполнении команд **ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ ПЕРЕНОС** и **ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ НЕТ ПЕРЕНОСА** проверяется значение разряда переноса регистра состояния. Обычно эти команды выполняются после команд арифметической обработки.

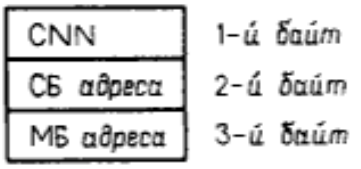
ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ МИНУС

CN,Адрес Если N = 1, то СК → Стек
Старший байт адреса → Старший байт счетчика команд
Младший байт адреса → Младший байт счетчика команд



ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ НЕ МИНУС

CNN,Адрес Если N = 0, то СК → Стек
Старший байт адреса → Старший байт счетчика команд
Младший байт адреса → Младший байт счетчика команд



При выполнении команд **ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ МИНУС** и **ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ НЕ МИНУС** проверяется значение разряда отри-

цательного результата в регистре состояния. Примечание: команду ПЕРЕХОД, ЕСЛИ НЕ МИНУС можно рассматривать как команду ПЕРЕХОД, ЕСЛИ ПЛЮС.

ВОЗВРАТ ИЗ ПОДПРОГРАММЫ
RET Стек → СК

RET

При выполнении команды ВОЗВРАТ ИЗ ПОДПРОГРАММЫ микропроцессор извлекает из стека значение счетчика команд, при котором произошел выход из программы предыдущего уровня, и загружает это значение в счетчик команд. Команде ВОЗВРАТ ИЗ ПОДПРОГРАММЫ ничего не сообщается относительно того, какого уровня вложения выполняется подпрограмма и сколько раз происходил вызов этой подпрограммы. В указателе стека такой информации также не содержится. Выполняя команду ВОЗВРАТ ИЗ ПОДПРОГРАММЫ, микропроцессор просто возвращается к тому значению счетчика, которое было последним загружено в стек.

Таким образом, микропроцессор никак не может определить, не являются ли данные, находящиеся в стеке и в указателе стека, неверными. Вот почему случайное искажение содержимого стека приводит к неправильному функционированию микропроцессорной системы.

На рис. 10.10 приведена модификация программы, представленной ранее на рис. 8.24. Модифицированная часть программы предназначена для отрицательного приращения содержимого счетчика. В данном случае она составлена таким образом, чтобы вместо множества команд, служащих для отрицательного приращения и проверки содержимого счетчика, использовать команды вызова подпрограмм.

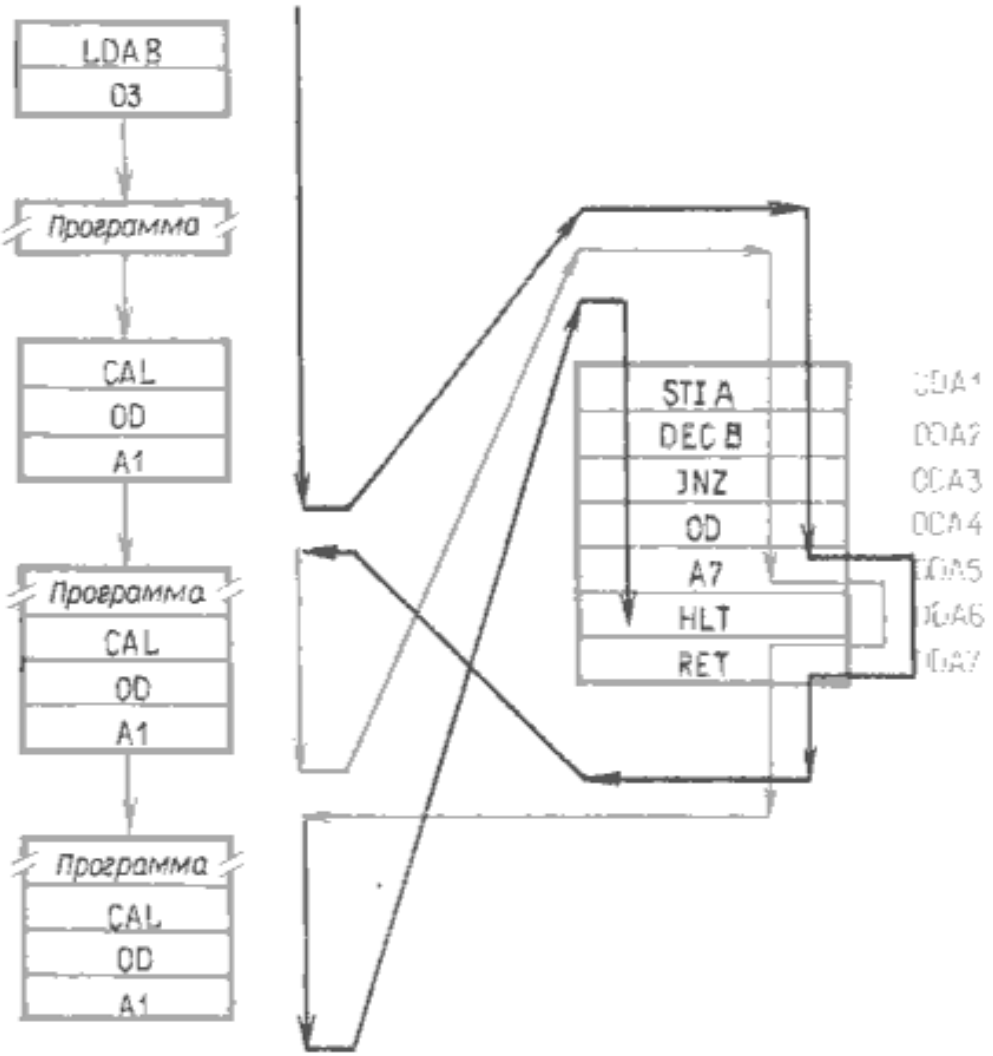


Рис. 10.10. Модификация программы, приведенной на рис. 8.24, после замены множества команд, предназначенных для отрицательного приращения и проверки значения счетчика, на команды вызова подпрограмм.

Выполнение программы начинается с прямой загрузки числа 03 в регистр B. Здесь следует ряд действий, реализующих главную программу. Когда эта часть программы заканчивается, происходит вызов подпрограммы. В данном случае для этой цели используется команда ВЫЗОВ ПОДПРОГРАММЫ БЕЗУСЛОВНЫЙ. Эта команда является для микропроцессора указанием на то, что необходимо использовать подпрограмму, показанную в правой части рис. 10.10. При входе в подпрограмму выполняется ее первая команда, которая представляет собой команду ЗАПИСЬ В ПАМЯТЬ КОСВЕННАЯ. По этой команде данные из аккумулятора записываются в область памяти, на которую указывает регистровая пара BC.

По следующей команде подпрограммы осуществляется отрицательное приращение содержимого регистра B. Далее следует команда ПЕРЕХОД, ЕСЛИ НЕ НУЛЬ. Если в результате отрицательного приращения содержимого регистра B разряд нулевого результата в регистре состояния не устанавливается в 1, то при выполнении команды ПЕРЕХОД, ЕСЛИ НЕ НУЛЬ осуществляется переход. В этом случае следующей является команда ВОЗВРАТ ИЗ ПОДПРОГРАММЫ, потому что адрес именно этой команды содержится во втором и гретьем байтах команды JNZ. Конечно, если после команды отрицательного приращения содержимое аккумулятора приобретает нулевое значение, то переход не происходит. (Осуществляется выборка команды перехода, но сам переход не реализуется.) Следующей в этом случае выполняется очередная по порядку команда – команда ОСТАНОВ.

Если переход осуществляется, то выполнение программы, которая произвела вызов подпрограммы, возобновляется с того места, на котором оно было прервано. Таким образом, в качестве очередной будет выполняться команда, следующая за командой ВЫЗОВ ПОДПРОГРАММЫ БЕЗУСЛОВНЫЙ. Выполнение главной программы продолжается до тех пор, пока опять не встретится команда ВЫЗОВ ПОДПРОГРАММЫ БЕЗУСЛОВНЫЙ. При этом во второй раз реализуется уже использованная ранее подпрограмма. Описанный процесс продолжается до тех пор, пока не будет прерван командой ОСТАНОВ. Окончательное прекращение выполнения программы явится результатом реализации команды ОСТАНОВ, входящей в состав подпрограммы.

Проследить действия, выполняемые программой после загрузки числа 03 в регистр B, можно с помощью линий со стрелками (см. рис. 10.10). Как можно видеть, вызов подпрограммы происходит троекратно. Прекращается выполнение программы командой ОСТАНОВ, входящей в подпрограмму. Обратите внимание на то, что упомянутая команда не обязательно должна присутствовать в главной программе.

Подпрограммы служат для реализации любого часто повторяющегося набора шагов программы. Так, например, в процессе решения сложной математической задачи могут иметь место многократные обращения к пакету программ обработки с повышенной точностью чисел с плавающей точкой. Часто представляются в виде подпрограмм программы, служащие для осуществления задержки, а также программы ввода данных, анализа данных клавиатур и дисплеев и других часто используемых устройств обработки информации.

Задания для самопроверки

10. Одно из преимуществ подпрограмм заключается в том, что их применение позволяет уменьшить длину программы. Какой размер (в байтах) должна иметь подпрограмма, чтобы такая экономия была достигнута? Почему?

11. Измените подпрограмму, приведенную на рис. 10.10, таким образом, чтобы команды ВОЗВРАТ ИЗ ПОДПРОГРАММЫ и ОСТАНОВ поменялись местами. Объясните работу полученной таким образом подпрограммы и действия, выполняемые входящими в нее командами.

12. Имеется программа длиной 276 команд, из которй с помощью команды ВЫЗОВ ПОДПРОГРАММЫ БЕЗУСЛОВНЫЙ производится десятикратное обращение к подпрограмме, состоящей из 25 команд. а) Быстрее или медленнее выполняется программа в этом случае, а не в том, когда подпрограмма входила бы

в главную программу в качестве составной части? б) Как соотносится размер данной программы с размером программы, содержащей 301 команду и включающей ту же подпрограмму? Предполагается, что и в подпрограмме, и в программе, составленной без подпрограммы, полностью используются одни и те же 25 команд.

13. Предположим, что применяется 8-разрядный микропроцессор, в котором имеется 16-разрядный счетчик команд. Вы написали программу, в которой предусмотрено пятиуровневое вложение подпрограмм. а) Объясните, что это означает. б) Какой размер (в байтах) должен иметь при этом стек?

14. В большинстве 8-разрядных микропроцессоров, снабженных памятью объемом 65К, используется 16-разрядный указатель стека. Стек соответственно может иметь размер до 65 536 байт. Конечно, на практике такой большой стек никогда не понадобится. Зачем нужен указатель стека такой длины, если размер стека никогда не достигает указанной величины?

15. Предположим, что при разработке подпрограммы четвертого уровня вложения (см. п. 13) допущена ошибка. Подпрограмма записывает данные в третий байт, считая от верхушки стека. Чревата ли последствиями такая ошибка, и если да, то какими? Каким образом могут проявиться эти последствия?

16. Имеется микропроцессорная система, в которой вычитание реализуется за счет использования дополнительного кода. К системе подключен индикатор, обеспечивающий отображение чисел со знаком плюс или минус. Для индикации отрицательных чисел результата необходимо разработать подпрограмму, с помощью которой отображается знак минус и формируется дополнительный код предварительно полученного результата. Какую команду следует использовать в данном случае для вызова подпрограммы и почему?

17. Объясните различия между командами перехода и вызова подпрограмм.

18. Почему некоторые программисты называют команду JNN командой ПЕРЕХОД, ЕСЛИ ПЛЮС?

19. Что представляют собой команды JZ и JE? Объясните происхождение мнемонического обозначения JE.

Упражнения

10.1. Объясните, почему при переходе к области памяти с меньшим номером образуется программный цикл.

10.2. В чем состоит различие между командами условного перехода?

10.3. Во всех командах перехода, используемых микропроцессором, применяется прямая адресация. Что это означает?

10.4. В каких ситуациях команда условного перехода может быть выбрана из памяти, но реализована не полностью?

10.5. В чем заключается различие между выполнением команд безусловного и условного переходов?

10.6. Какой из нижеперечисленных команд эквивалентна команда ПЕРЕХОД, ЕСЛИ РАВНО:
а) ПЕРЕХОД, ЕСЛИ ПЕРЕНОС,
б) ПЕРЕХОД, ЕСЛИ ПЛЮС,
в) ПЕРЕХОД, ЕСЛИ НЕ МИНУС,
г) ПЕРЕХОД, ЕСЛИ НУЛЬ?

10.7. Какой из нижеперечисленных команд эквивалентна команда ПЕРЕХОД, ЕСЛИ НЕ МИНУС:
а) ПЕРЕХОД, ЕСЛИ ПЕРЕНОС,
б) ПЕРЕХОД, ЕСЛИ ПЛЮС,
в) ПЕРЕХОД, ЕСЛИ МИНУС,
г) ПЕРЕХОД, ЕСЛИ НУЛЬ?

10.8. Какие команды используются для реализации блок-схемы алгоритма, представленной на рис. 10.11?

10.9. Какой блок следует заменить в блок-схеме алгоритма, приведенной на рис. 10.11, если поменять местами ответы «Да» и «Нет»?

10.10. Изменится ли блок-схема, представленная на рис. 10.11, если содержимое блока обработки заменить на «Прибавление 1 к аккумулятору»?

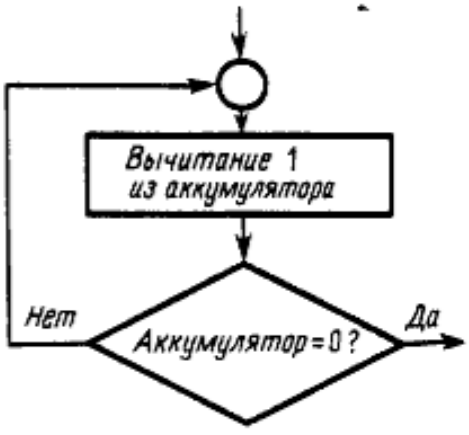


Рис. 10.11. Блок-схема алгоритма к упр. 10.8.

10.11. Фрагмент программы, рассмотренный в упр. 10.8, часто называют подпрограммой задержки. Предположим, что цикл тактирования микропроцессора составляет 1 мкс. Какая команда должна предшествовать такой подпрограмме, и какие данные должны содержаться в этой команде, если необходимо обеспечить задержку продолжительностью 1 мс?

10.12. В чем заключается основное различие между командами перехода и командами вызова подпрограмм?

10.13. Какая из нижеперечисленных проверок осуществляется при обращении к подпрограмме с помощью команды условного вызова:

а) равенства 0 всех разрядов аккумулятора;

б) достижения содержимым регистра определенного значения после отрицательного приращения;

в) значений разрядов регистра состояния;

г) содержимого регистра команд?

10.14. Какое из указанных ниже действий выполняется при вызове одной подпрограммы из другой подпрограммы:

а) команда JNE; б) вложение подпрограмм;

в) ветвление программы; г) проверка условий?

10.15. Какое из приведенных ниже действий осуществляется с помощью команды ВОЗВРАТ ИЗ ПОДПРОГРАММЫ:

а) загрузка данных в стек; б) извлечение данных из стека; в) начальная установка стека; г) нарушение работы стека?

10.16. Какой максимальный объем (в байтах) может иметь стек микропроцессора:

а) 8; б) 16; в) 128; г) 65 536?

10.17. Какое из указанных ниже действий может произойти при изъятии данных из микропроцессорного стека, если его содержимое подверглось изменениям:

а) извлечение из стека;

б) загрузка в стек;

в) вложение подпрограмм;

г) нарушение работы программы?

10.18. При каком значении аккумулятора происходит вызов подпрограммы в процессе выполнения команды ВЫЗОВ ПОДПРОГРАММЫ, ЕСЛИ НЕ МИНУС:

а) положительном;

б) не положительном;

в) 1 в старшем разряде;

г) 1 в старшем и младшем разрядах?

10.19. Объясните, какие ограничения накладываются на адреса областей памяти, которые могут быть использованы при выполнении команд условного перехода.

10.20. Одним из первых 8-разрядных микропроцессоров появился микропроцессор 8008 фирмы Intel. Его стек состоит из семи 14-разрядных внутренних регистров; счетчик команд также 14-разрядный. Какие ограничения накладываются при этом на глубину вложения подпрограмм и объем адресуемой памяти?

10.21. Сопоставьте известные вам характеристики микропоцессора 8008 фирмы Intel с аналогичными характеристиками современного 8-разрядного микропроцессора.

10.22. В 16-разрядной микро-ЭВМ LSI-11 фирмы Digital Equipment (DEC) один из 16-разрядных регистров общего назначения используется в качестве указателя стека. При этом старший бит адреса содержит информацию о том, происходит обращение к старшему или младшему байту адресуемой области памяти. Таким образом, в микро-ЭВМ LSI-11 могут быть адресованы 32 768 слов памяти, т.е. и объем стека ограничивается 32 768 16-разрядными словами. Является ли это ограничение существенным в сопоставлении со способностью 8-разрядного микропроцессора адресовать до 65 536 областей памяти и иметь, следовательно, стек размером до 65 536 байт?

Ответы на вопросы заданий для самопроверки

1. См. рис. 10.12. Необходимые команды:
   CPI, Данные
   JZ (Равно)
   JNN (Больше чем)
2. Начальная установка счетчика производится с помощью команды LDA B или другой команды, служащей для загрузки регистра. Уменьшение содержимого счетчика на 10 выявляется

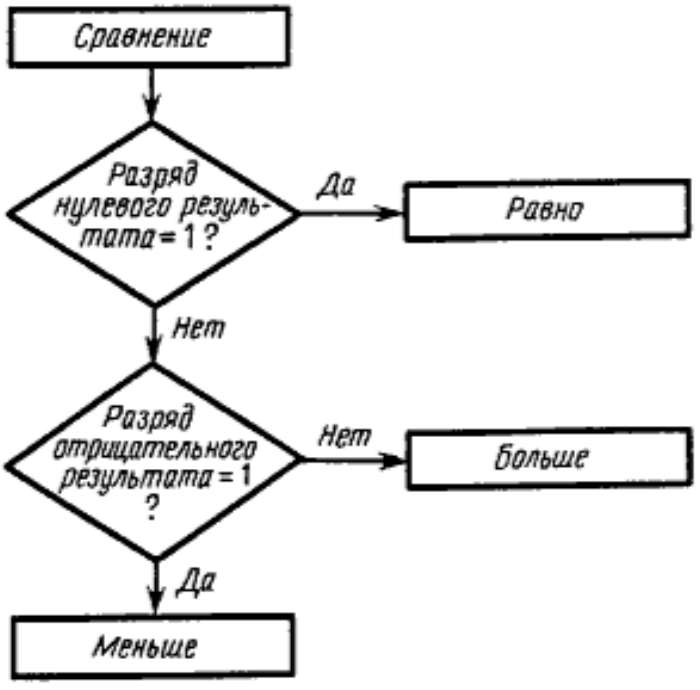


Рис. 10.12. Блок-схема алгоритма к ответу на вопрос п. 1 заданий для самопроверки.

с помощью команды ПЕРЕХОД, ЕСЛИ НЕ НУЛЬ. Блок-схема алгоритма приведена на рис. 10.13. Если применить команду JZ, то блок проверки условия имеет вид, показанный на рис. 10.14.

3. а) 006А. б) Аккумулятор или регистр. Команда представляет собой ПЕРЕХОД, ЕСЛИ НЕ МИНУС. При очистке аккумулятора его старший разряд приобретает значение, равное 0. Данные при этом имеют положительную величину, и осуществляется переход. Программа имеет вид:

0129	CLA
012A	JNN
012B	00
012C	6A

4. а) 012D. б) Старший бит аккумулятора или регистра. Если старший разряд установлен в 1, переход не осуществляется. Очередная выполняемая в этом случае команда располагается в памяти сразу же за адресными байтами команды перехода.

5. а) JZ. б) СРАВНЕНИЕ.

6. Переход к команде, находящейся в области 00B9. См. рис. 10.15.

7.

Содержимое СК при обработке		
отрицательных данных	положительных данных	нулевых данных
0000	0000	0000
0002	0002	0002
0004	0004	0004
0007	0007	0002
000A	000A	
000D	000D	
0010	0013	
0011	0014	
0012	0015	
0013	0016	
0014	0017	
0015	001A	
0016	001B	
0017	001C	
001A	001D	
001B	001F	
001C	0020	
001D	0021	
001F	0022	
0020	0002	
0021		
0022		
0002		

8. См. рис. 10.16. Команды перехода, находящиеся в областях 0012, 0024 и 0032, должны представлять собой команды перехода к области 0000.

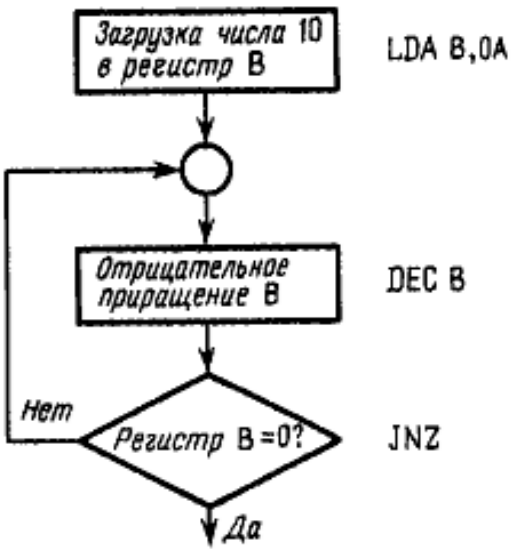


Рис. 10.13. Блок-схема алгоритма к ответу на вопрос п. 2 заданий для самопроверки.

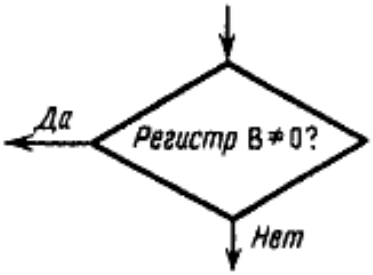


Рис. 10.14. Блок проверки условия к ответу на вопрос п. 2 заданий для самопроверки при использовании команды JZ.

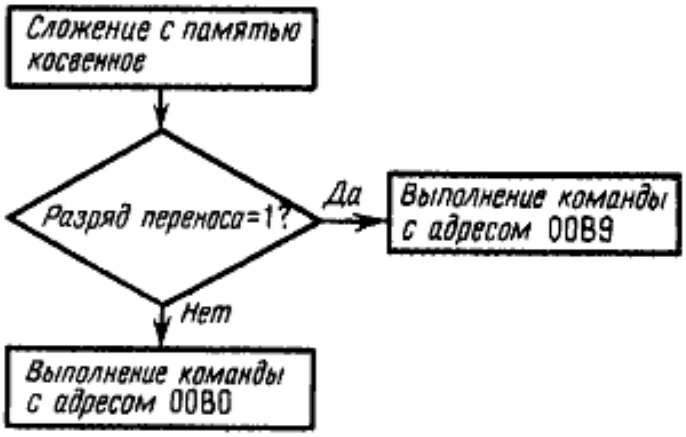


Рис. 10.15. Блок-схема программы к ответу на вопрос п. 6 заданий для самопроверки.

9. Добавляются команды с адресами 0035–003B. Счетчики (указатели), хранимые в областях 00FB, 00FC и 00FD, осуществляют соответственно подсчет количества малых, средних и больших чисел. Они имеют двойное назначение.

10. Не менее 4 байт, потому что длина команды вызова подпрограммы составляет 3 байт, а для

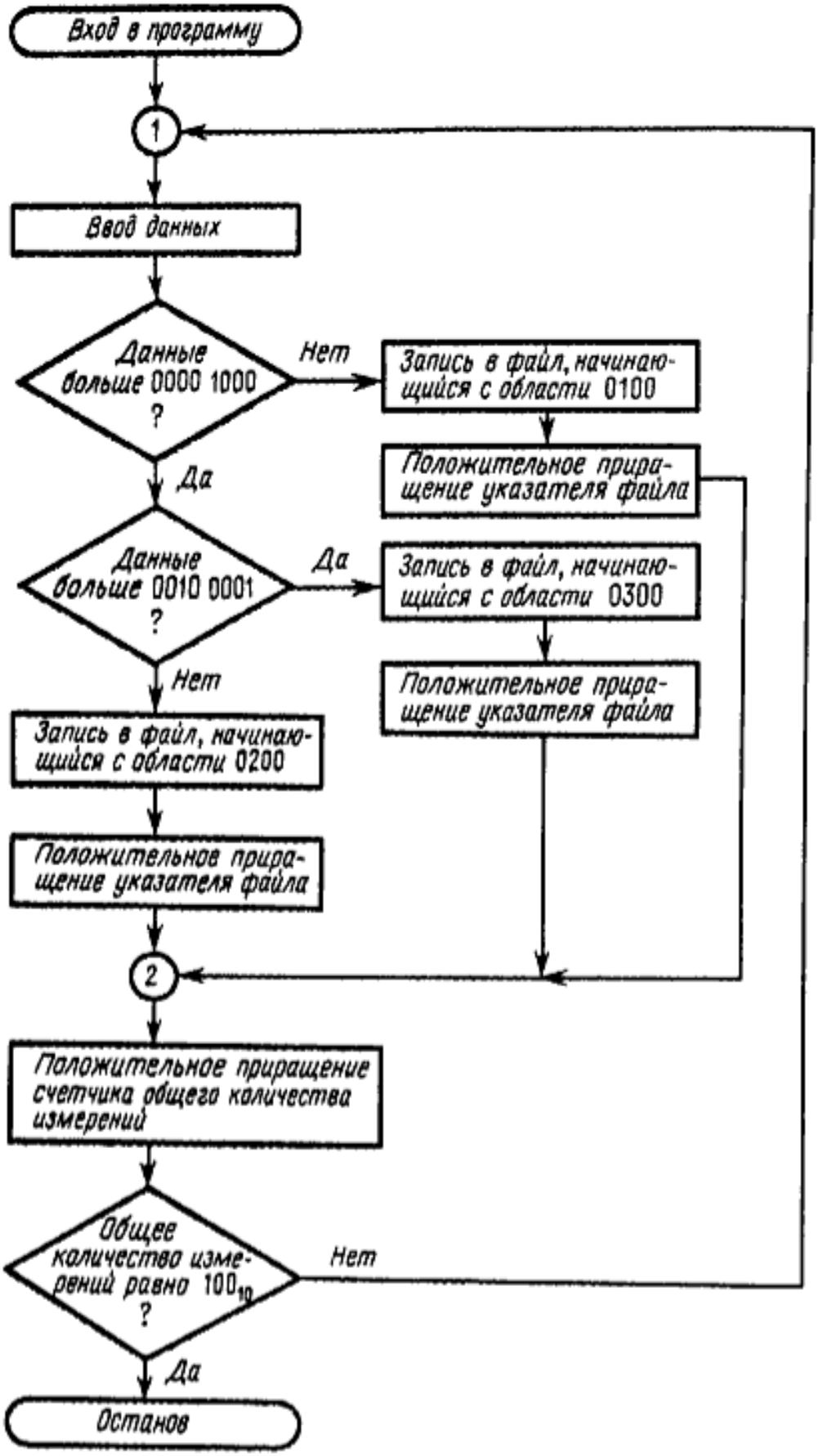


Рис. 10.16 а. Блок-схема алгоритма к ответу на вопрос п. 8 заданий для самопроверки.

работы с подпрограммами такая команда необходима.

11. Надо заменить команду JNZ на JZ. Команды ОСТАНОВ и ВОЗВРАТ ИЗ ПОДПРОГРАММЫ меняются при этом местами, так как переход к команде ОСТАНОВ должен теперь происходить, когда значение счетчика в результате отрицательных приращений становится равным 0.

12. а) При использовании подпрограммы время выполнения программы несколько возрастает,

Адрес области памяти	Содержимое области памяти	Комментарий
0000	LOA D	Начальная установка регистра D в 0
0001	00	
0002	LOA B	Загрузка в регистр B СБ указателя файла малых чисел
0003	01	
0004	IN	Ввод данных
0005	01	
0006	CMP I	Значение данных равно 00001000?
0007	08	
0008	JC	Если больше 0000 1000, то переход
0009	00	
000A	17	
000B	LDD C	Извлечение МБ указателя файла малых чисел
000C	00	
000D	FB	
000E	STI A	Запись данных в файл малых чисел
000F	INC C	Положительное приращение младшего байта указателя файла
0010	MOV A,C	Пересылка указателя файла в аккумулятор
0011	STA A	Запись увеличенного значения указателя файла на прежнее место
0012	00	
0013	FB	
0014	JMP	Формирование файла завершено, переход к окончанию программы
0015	00	

Рис. 10.16 б. Листинг программы к ответу на вопрос п. 8 заданий для самопроверки.

Адрес области памяти	Содержимое области памяти	Комментарий
0016	37	
0017	CMP I	Значение данных равно 0010 0001?
0018	21	
0019	JC	Если больше 0010 0001, то переход
001A	00	
001B	29	
001C	LDD C	Извлечение МБ указателя файла средних чисел
001D	00	
001E	FC	
001F	INC B	Формирование в регистре B СБ указателя файла средних чисел
0020	STI A	Запись данных в файл средних чисел
0021	INC C	Положительное приращение МБ указателя файла
0022	MOV A,C	Пересылка указателя файла в аккумулятор
0023	STA A	Запись увеличенного значения указателя файла на прежнее место
0024	00	
0025	FC	
0026	JMP	Формирование файла завершено, переход к окончанию программы
0027	00	
0028	37	
0029	LDD C	Извлечение МБ указателя файла больших чисел.
002A	00	
002B	FD	
002C	INC B	Формирование в регистре B СБ указателя файла средних чисел
002D	INC B	Формирование в регистре B СБ указателя файла больших чисел
002E	STI A	Запись данных в файл больших чисел
002F	INC C	Положительное приращение МБ указателя файла

Рис. 10.16б. Продолжение.

Адрес области памяти	Содержимое области памяти	Комментарий
0030	MOV A,C	Пересылка указателя файла в аккумулятор
0031	STA A	Запись увеличенного значения указателя файла на прежнее место
0032	00	
0033	FD	
0034	JMP	Формирование файла завершено, переход к окончанию программы
0035	00	
0036	37	
0037	INC D	Окончание. Прибавление 1 к содержимому регистра D
0038	CMP I	Значение $D = 100_{10}$?
0039	64	
003A	JNZ	Нет, продолжение приема данных
003B	00	
003C	02	
003D	HLT	Да, выход из программы

Рис. 10.16б. Продолжение.

так как в процессе выполнения необходимо 10 раз применить команду ВЫЗОВ ПОДПРОГРАММЫ. б) Если подпрограммы не использовать, то размер программы составит: 276 команд минус 10 команд ВЫЗОВ ПОДПРОГРАММЫ плюс 25 × 10, т. е. 250 команд, которые включаются в программу вместо подпрограммы и команд ее вызова. Таким образом, размер программы, в которой не используется подпрограмма, равен

$$276-10+250=516 \text{ команд.}$$

13. а) Это означает, что из первой подпрограммы производится вызов второй, из второй – третьей и т. д. б) Стек должен вмещать по 2 байт для каждой из четырех первых подпрограмм и для главной программы, т. е. иметь объем 10 байт.
14. При длине указателя стека, составляющей 16 разрядов, организацию стека можно начать с любой области памяти.
15. При этом неправильно определится точка возврата из второй подпрограммы. Весьма вероятно, что работа микропроцессора нарушится.
16. CN,Адрес. При реализации этой команды происходит вызов подпрограммы, если в результате выполнения предыдущей команды сформировано отрицательное число, которое для вывода на индикатор необходимо преобразовать в обратный код и подвергнуть положительному приращению.
17. Команды перехода только изменяют последовательность выполнения программы, а в командах вызова подпрограмм заложена возможность возврата к исходной точке главной подпрограммы.
18. Потому что при выполнении команды ПЕРЕХОД, ЕСЛИ НЕ МИНУС переход реализуется, если число положительное.
19. Это команды ПЕРЕХОД, ЕСЛИ НУЛЬ и ПЕРЕХОД, ЕСЛИ РАВНО. Команда ПЕРЕХОД, ЕСЛИ РАВНО – это по существу команда JZ, используемая после команды СРАВНЕНИЕ. Если при сравнении выявляется равенство сравниваемых значений, то устанавливается в 1 разряд нулевого результата.

Глава 11. Память

В предыдущих главах было рассмотрено множество систем, построенных на основе микропроцессоров. Каждая их них была снабжена памятью. Некоторые системы располагают памятью совсем небольшого объема, достаточной лишь для размещения программы и хранения нескольких байтов данных. Объем памяти других микропроцессорных систем таков, что полностью используются их адресные возможности.

В данной главе будут описаны аппаратные средства памяти. Мы выйдем за рамки представления о памяти лишь как о совокупности ячеек, в которые можно записывать данные и из которых можно их считывать. Будут рассмотрены оперативные и постоянные запоминающие устройства, реализованные как на статических, так и на динамических элементах. Будет выяснено, почему существуют эти два вида памяти, как они применяются, какими достоинствами и недостатками характеризуются.

Для организации памяти микропроцессорных систем применяются постоянные запоминающие устройства (ПЗУ) четырех типов, с особенностями которых мы также ознакомимся в данной главе. Наконец, будет рассмотрен особый способ быстрого обмена информацией с памятью, называемый *прямым доступом к памяти (ПДП)*.

11.1. Оперативные запоминающие устройства с произвольным доступом

Степень нашего знакомства с микропроцессором позволяет утверждать, что он должен располагать возможностями чтения команд, составляющих определенную программу, и запоминания данных, вырабатываемых этой программой. Таким образом, необходимо иметь память, позволяющую читать и записывать информацию.

Для архитектуры современных микропроцессоров характерно наличие единого адресуемого пространства памяти, которое называется *основной памятью*. По крайней мере какая-то часть основной памяти должна представлять собой *оперативную память*, т. е. память с возможностью как чтения областей памяти, так и записи в них информации.

В некоторых микропроцессорных системах общего назначения почти все пространство памяти является оперативным. Команды программы записывают в нее с помощью операции записи в память. Впоследствии в ходе выполнения программы микропроцессор считывает из нее команды. Разумеется, данные также записываются

в области памяти и считываются из них. Большинство запоминающих устройств микропроцессорных систем представляет собой оперативную память.

Кроме того, подавляющая часть запоминающих устройств микропроцессоров является *устройством с произвольным доступом*. Возможность произвольного доступа к памяти настолько характерна для микропроцессорных систем, что иногда, не задумываясь, называют всю память микропроцессорной системы памятью с произвольным доступом, а это не всегда справедливо.

Если память не обеспечивает возможности произвольного доступа, то ее относят к *памяти с последовательным доступом*. Этот тип памяти редко используется в качестве основной памяти микропроцессорной системы. При работе с такой памятью для обращения к нужной области необходимо предварительно обратиться ко всем областям, лежащим между той, к которой произошло обращение в данный момент, и требуемой областью. В последовательной форме хранится информация на магнитной ленте. Память такого рода применяется для размещения больших массивов информации. Последовательный доступ приемлем, когда речь идет о запоминании значительных объемов данных, время обращения к которым не является критичным.

При работе с памятью часто говорят о *времени доступа к памяти* и *времени цикла памяти*. Оба этих параметра являются характеристиками *быстродействия* памяти. Они тесно связаны между собой, т.е. чем меньше время доступа к памяти, тем короче время цикла памяти.

Одна из разновидностей параметра времени доступа служит для указания на то, какое время необходимо для вывода информации из памяти на шину данных после адресации нужной области памяти. Она называется *временем доступа при чтении* или *временем чтения данных из памяти*. Другая разновидность указанного параметра – *время доступа при записи* – это время, необходимое для записи данных в адресуемую область.

Время доступа определяется организацией памяти и скоростью работы схем, на которых построена память. Например, память на интегральных схемах может иметь время доступа 200 нс. Это означает, что время от адресации любой области памяти до поступления данных на выходную шину памяти составляет 200 нс.

Наряду с этим при работе с памятью на магнитной ленте время доступа может иметь довольно значительную величину. В данном случае оно по существу зависит от местоположения адресуемой области на ленте. Если адресуемая область находится близко к началу ленты, время доступа относительно невелико; если же адресуемая область расположена ближе к концу ленты, то время доступа сильно возрастает. Говоря об устройствах с последовательным доступом, обычно имеют в виду *среднее время доступа*. Так, если для полной перемотки ленты требуется 40 с, то среднее время доступа составляет 20 с.

Быстродействие памяти характеризуется, кроме того, временем цикла. *Время цикла* – это наименьший интервал времени, который может иметь место между двумя обращениями к памяти. Он зависит не только от характеристик, которые свойственны памяти, но и от других временны́х параметров микропроцессорной системы. Эти параметры могут быть такими, что очередное использование памяти возможно не раньше чем по истечении определенного промежутка времени после предыдущего обращения к ней.

При описании памяти микропроцессорных систем часто используется следующая ее характеристика: *энергозависимая* или *энергонезависимая память*. В энергонезависимой памяти данные при нарушениях в работе системы питания не разрушаются, а в энергозависимой – разрушаются.

Очевидно, что, поскольку микропроцессор не может выполнять никаких действий при отсутствии команд, он должен иметь какую-то энергонезависимую память. Однако объем такой энергонезависимой памяти не обязательно должен быть большим. Необходимо, чтобы его хватило лишь для запоминания короткой программы. Выполняя эту программу, микропроцессор переписывает в основную память остальные команды из энергонезависимой внешней памяти большого объема, такой, как устройство хранения информации на магнитной ленте

или на магнитном диске. Как будет показано ниже, подобные энергонезависимые программы обычно хранятся в различного типа ПЗУ.

Очевидно, что команды программы также необходимо запоминать в какой-либо энергонезависимой памяти. В противном случае перед каждым прогоном программы пришлось бы воссоздавать ее заново. Одним из способов сохранения программы в энергонезависимой памяти является запись ее на бумаге.

Устройства хранения информации на магнитных лентах и дисках относятся к энергонезависимым. Энергонезависимой является также память на магнитных сердечниках. Полупроводниковая же оперативная память, если для нее не предусмотрен резервный аккумуляторный источник питания, энергозависима.

Значительная часть полупроводниковых устройств основной памяти снабжается сейчас резервным аккумуляторным питанием, благодаря чему они становятся *временно* энергонезависимыми, так как способны сохранять информацию в периоды кратковременных отказов источников питания. В больших ЭВМ память продолжает правильно функционировать на батарейном питании лишь в течение 15–20 мин. В то же время при отказе источника питания памяти небольших микро-ЭВМ нормальная работа последней может продолжаться до нескольких дней. Вообще говоря, резервное аккумуляторное питание не обеспечивает постоянной энергонезависимости, потому что мощность, потребляемая полупроводниковыми запоминающими устройствами, настолько велика, что для поддержания работоспособности памяти в течение сколько-нибудь продолжительных интервалов времени необходимы весьма громоздкие аккумуляторы.

Когда-то самым распространенным видом памяти были запоминающие устройства на магнитных сердечниках, в которых каждый бит информации хранится в небольшом тороидальном постоянном магните. Постоянный магнит намагничивается в одном из двух направлений. Направление намагниченности задается направлением тока, протекающего по пронизывающим память проводникам. В настоящее время магнитные запоминающие устройства применяются редко, так как они велики по габаритам, дороги, потребляют большой ток и рассеивают значительное количество тепла. Современные ЭВМ, как правило, содержат вместо устаревших магнитных запоминающих устройств устройства памяти полупроводникового типа.

Для реализации полупроводниковой памяти применяются два основных вида технологии – те же самые, что и для изготовления других цифровых интегральных схем: биполярная и МОП (металл–окисел–полупроводник)-технологии.

Память на биполярных транзисторах в микропроцессорных системах применяется редко. Достоинством такой памяти является очень короткое время доступа. В то же время по сравнению с памятью на МОП-транзисторах ей присущи некоторые недостатки. Она потребляет значительную мощность и допускает размещение меньшего числа битов при одинаковых размерах кристалла. Процесс изготовления биполярных интегральных схем гораздо более сложен, чем МОП-приборов. По этим причинам память на биполярных транзисторах оказывается значительно дороже памяти на МОП-приборах и применяется лишь в тех случаях, когда необходимо обеспечить высокое быстродействие.

На сегодняшний день для микропроцессорных систем наиболее характерно использование памяти на МОП-транзисторах. Существуют два способа построения интегральных схем памяти на МОП-технологии, в соответствии с которыми память на МОП-структурах может быть *статической* или *динамической*. Статическая память проще с точки зрения организации, что особенно явно проявляется при запоминающих устройствах небольшого объема. Управлять работой статической памяти также легче. Интегральные схемы, применяемые для построения динамической памяти, относительно дешевы, но в состав такой памяти входит много вспомогательных интегральных схем. Кроме того, содержимое динамической памяти необходимо периодически регенерировать. На динамическом принципе строятся запоминающие устройства большого объема.

Технология изготовления и статических, и динамических запоминающих устройств

постоянно совершенствуется. Это означает, что на кремниевой подложке одного размера размещается память все большего объема. Как следствие этого постоянно снижается стоимость памяти. При существующей тенденции удешевления памяти наблюдается постоянное возрастание объема памяти, требуемого для построения микропроцессорных систем, т. е. одни и те же задачи решаются с использованием памяти большего объема. Конечно, при этом изменяются и способы решения задач.

Знакомясь, например, в гл. 5 с программированием, мы видели, что существует два основных вида программ: на языках ассемблера и на языках высокого уровня. Программы на языках ассемблера относительно компактны, но их разработка занимает значительное время. Написание программ на языках высокого уровня требует меньших затрат времени, но для их размещения требуется гораздо больший объем памяти, чем для программ на языках ассемблера.

Легко понять, что быстрое снижение стоимости полупроводниковой памяти определяет все более широкое использование языков высокого уровня при разработке микропроцессорных систем. Если стоимость дополнительных затрат памяти меньше, чем дополнительная стоимость разработки программ, связанная с использованием языка ассемблера, то разработчик предпочитает прибегнуть к дополнительным затратам памяти.

В некоторых микропроцессорных системах начинают находить применение еще два вида полупроводниковой памяти. Это память на *приборах с зарядовой связью* (ППЗС) и память на *пузырьковых магнитных доменах* (ППМД). Оба этих типа памяти ориентированы на последовательный доступ. Как и другие устройства последовательного доступа, они имеют невысокое быстродействие, однако позволяют реализовать запоминающие устройства значительно большего объема, чем устройства памяти на биполярных и МОП-транзисторах.

Достоинством ППЗС является чрезвычайно малое потребление мощности. Кроме того, они очень просты в применении. Память на ППЗС энергозависима, но малая величина потребляемой ею мощности делает возможным сохранение в ней информации с помощью резервных аккумуляторов при отказе основного источника питания на длительное время.

Для памяти на пузырьковых магнитных доменах требуется гораздо большее число вспомогательных схем, чем для памяти на приборах с зарядовой связью. В то же время ППМД является в полном смысле слова энергонезависимой. Таким образом, этот тип памяти позволяет реализовать энергонезависимые запоминающие устройства большого объема, для которых не требуются сложные механические лентопротяжные механизмы и дисководы.

Задания для самопроверки

1) К какому из нижеперечиоленных типов памяти относится основная память микропроцессорной системы:

а) память с произвольным доступом;

б) оперативная память;

в) полупроводниковая память;

г) память, обладающая всеми названными характеристиками?

2. Быстродействие памяти часто характеризуется временем, необходимым для поступления данных на шину данных микроЭВМ после того, как произошла адресация памяти. Как называется эта характеристика:

а) временем цикла, б) последовательным доступом, в) временем доступа или г) произвольным доступом?

3. Какое из нижеперечисленных устройств представляет собой запоминающее устройство с произвольным доступом:

а) память на магнитной ленте;

б) постоянная память;

в) регистр сдвига на приборах с зарядовой связью?

4. Память на пузырьковых магнитных доменах используется в составе интеллектуальных терминалов, построенных на основе микропроцессоров. Она служит для хранения данных, необходимых терминалу, находящемуся в определенном месте, для решения определенной задачи. Благодаря каким из указанных здесь характеристик находит применение этот вид памяти в подобных ситуациях:

а) энергонезависимости;

б) низкому потреблению мощности;

в) энергозависимости;

г) сверхмалым размерам?

5. Типичная микропроцессорная система конца 70-х годов имела основную память объемом 12К байт. Часто в качестве таковой использовалась статическая МОП-память. Даже в настоящее время память, имеющая емкость 65К байт, считается большой. Какие из нижеперечисленных устройств следует применять для ее реализации:

а) запоминающие устройства на пузырьковых магнитных доменах;

б) динамические запоминающие устройства;

в) запоминающие устройства на магнитных сердечниках;

г) запоминающие устройства на магнитной ленте?

6. Запоминающие устройства с последовательным доступом обычно применяются в микропроцессорных системах для обеспечения а) произвольного доступа, б) малого времени доступа или в) хранения больших объемов информации?

7. Энергонезависимость современных ОЗУ, выполненных на МОП-приборах, достигается за счет а) наличия резервного источника питания, б) применения памяти на сердечниках, в) малого рассеяния мощности или г) использования ППЗС?

11.2. Статические и динамические запоминающие устройства

Статические запоминающие устройства являются, пожалуй, наиболее распространенным видом памяти микропроцессорных систем. Большинство статических запоминающих устройств реализуется на основе МОП-технологии, но существуют и статические запоминающие устройства на биполярных схемах. В этом разделе будут кратко рассмотрены способы построения МОП и биполярных статических запоминающих устройств, а также типичная организация интегральных схем памяти.

На рис. 11.1 показана упрощенная схема *биполярной ячейки памяти*. В такой ячейке хранится 1 бит информации. Ячейка реализована с использованием технологии многоэмиттерной транзисторно-транзисторной логики (ТТЛ). Как следует из схемы, ячейка памяти представляет собой не что иное, как обычный триггер. Этот триггер может быть установлен либо в состояние 1, либо в состояние 0 (сброс). Если триггер установлен в 1, то это значение сохраняется в нем до тех пор, пока не будет произведен сброс или не будет выключено питание.

Подобные ячейки памяти объединяются в матричную структуру – размещаются по строкам и столбцам. Пример такой организации ячеек приведен на рис. 11.2 для типичного кристалла памяти объемом

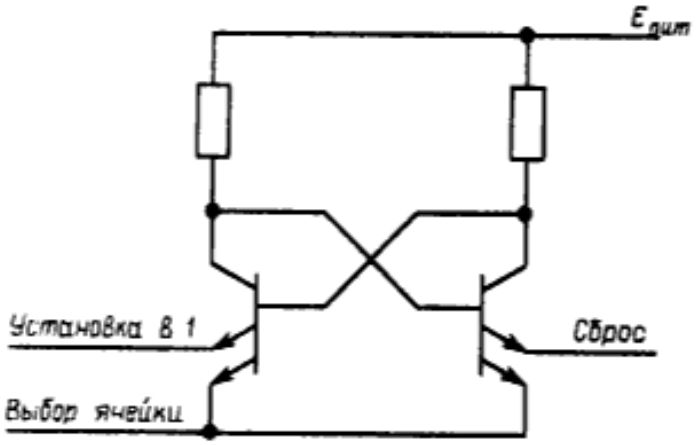


Рис. 11.1. Ячейка памяти на биполярных транзисторах. (Для образования триггера два многоэмиттерных транзистора охвачены обратными связями. Установка в 1 и сброс триггера осуществляются подачей сигналов на соответствующие эмиттеры. При обращении к ячейке по линии выбора образуется низкоимпедансный выходной сигнал допустимого уровня.)

4096 бит. Подобные структурные схемы одинаковы как для статической памяти на биполярных и МОП-структурах, так и для динамической памяти.

Каждый кристалл (интегральная схема) снабжен 12 адресными линиями A0–A11, которые поступают на дешифраторы адресов строк и столбцов. Первые шесть адресных линий A0–A5 поступают на дешифратор столбца, который преобразует шесть разрядов адреса в сигнал, указывающий один из 64 столбцов. Дешифратор строки преобразует шесть остальных разрядов адреса в сигнал, указывающий одну из 64 строк. Выбираемая ячейка памяти находится на пересечении выбранных строки

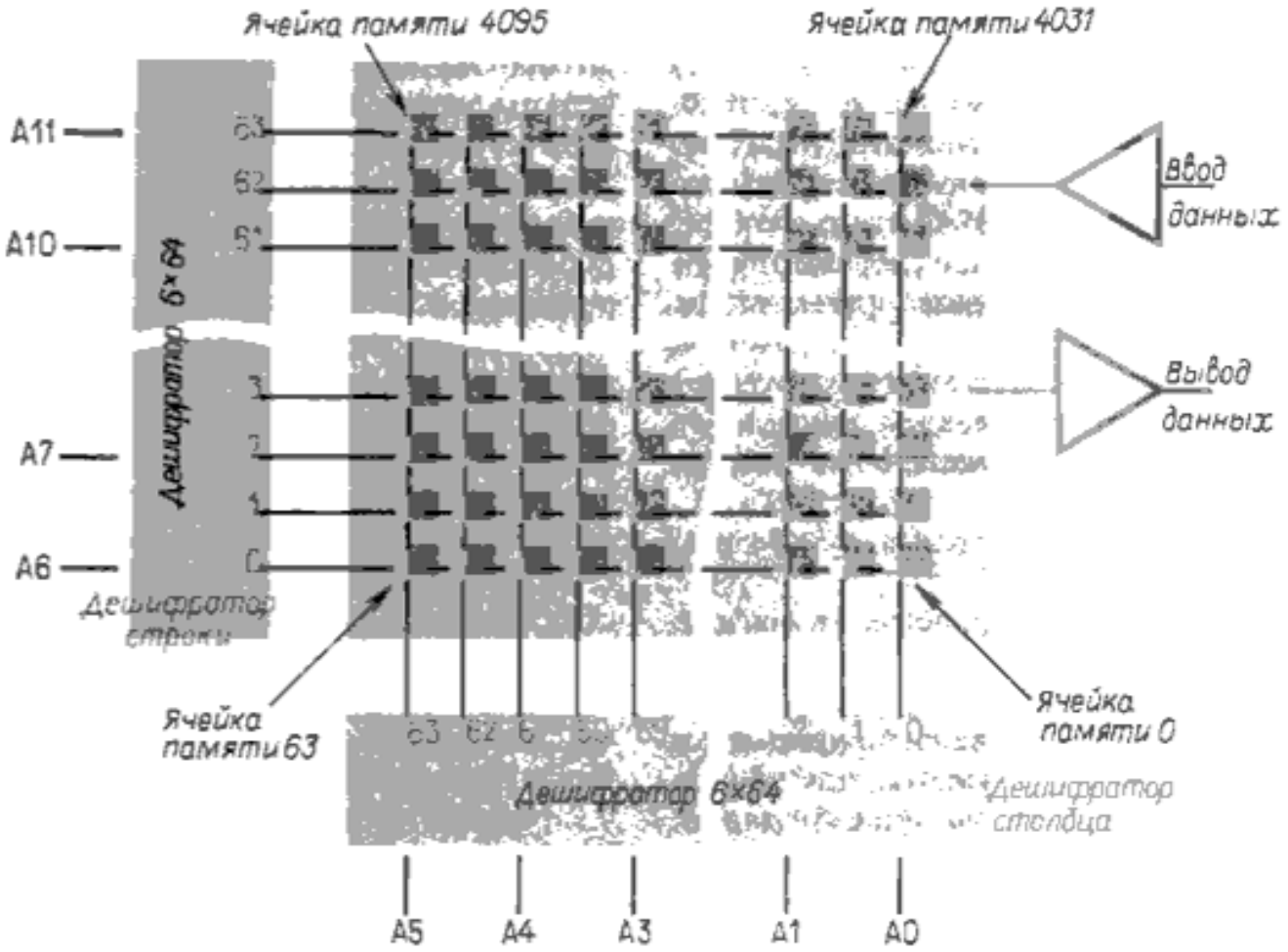


Рис. 11.2. Адресация памяти объемом 4096 бит по строкам и столбцам. (Каждая ячейка снабжена входами, с помощью которых осуществляется ее выбор подачей сигналов на соответствующие линии строки и столбца. Линии ввода и вывода данных соединены со всеми ячейками.)

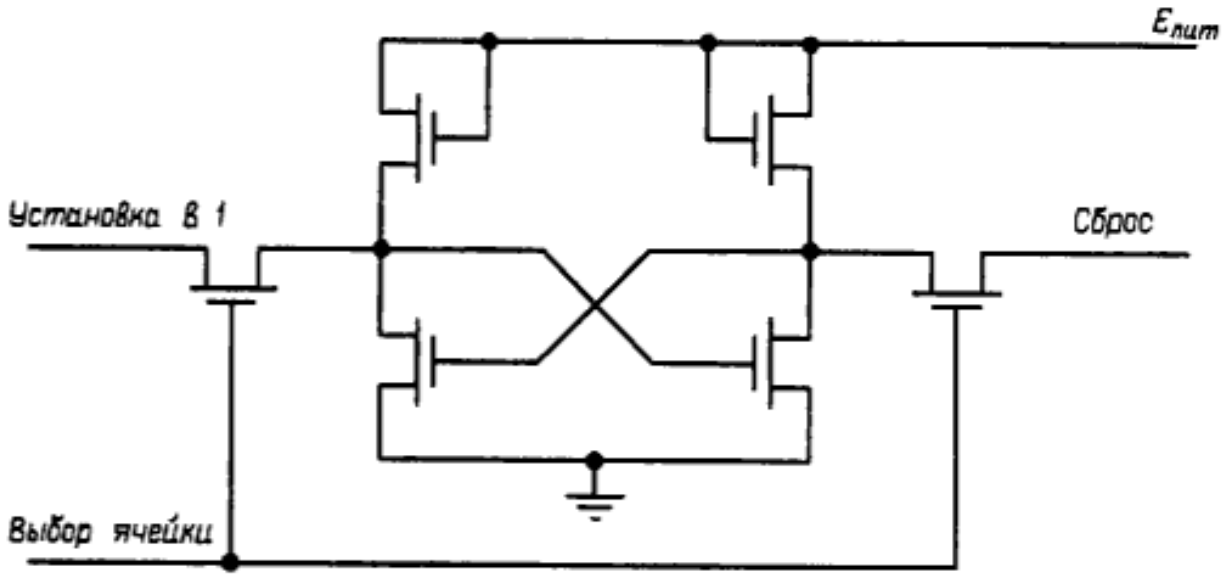


Рис. 11.3. Ячейка статической памяти на МОП-транзисторах.

и столбца. Простой арифметический подсчет показывает, что всего имеется $64 \times 64 = 4096$ возможных пересечений. Таким образом, рассмотренный метод адресации позволяет выбирать одну из находящихся на кристалле 4096 ячеек памяти. После того как ячейка выбрана, можно либо записывать в нее информацию, либо считывать.

На рис. 11.3 приведена упрощенная схе-

ма ячейки статической МОП-памяти. В сущности она также представляет собой триггер. Как и в большинстве устройств на МОП-структурах, в качестве нагрузочных сопротивлений здесь используются МОП-транзисторы с постоянно смещенными затворами. Для ввода и вывода информации из ячейки служит еще одна пара МОП-транзисторов.

Как и в случае биполярной ячейки памяти, запоминание двоичной информации в ячейке статической МОП-памяти обеспечивается за счет перекрестных обратных связей между двумя логическими элементами. Значение 1 или 0 сохраняется в ячейке, пока на нее подано напряжение питания.

Ячейка динамической МОП-памяти, показанная на рис. 11.4, значительно проще. С помощью вдвое меньшего количества транзисторов, чем в ячейке статической памяти, реализованы те же функции. Очевидно, интегральная схема динамической памяти может содержать вдвое большее количество битов, чем статической памяти, при одинаковых размерах кристаллов.

Ячейка динамической МОП-памяти строится не в виде триггера. Показанный на схеме конденсатор представляет собой входную емкость МОП-транзистора. Эта небольшая емкость играет в ячейке динамической МОП-памяти роль запоминающего элемента. Хранение единицы двоичной информации осуществляется путем удержания заряда емкости, соответствующего логической 1, в течение нескольких

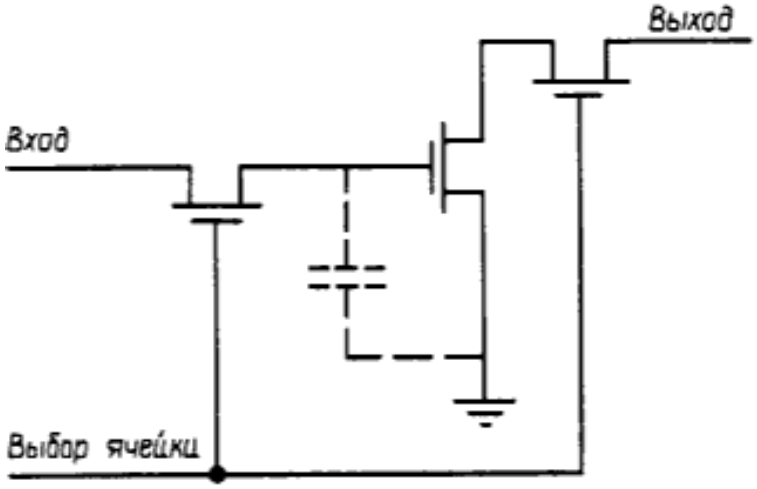


Рис. 11.4. Ячейка динамической памяти на МОП-транзисторах. (При заряде этого конденсатора в ячейке хранится 1, при отсутствии заряда – 0.)

миллисекунд. По истечении этого времени необходимо выполнить перезапись данных в ячейку. Отсутствие заряда на конденсаторе соответствует хранению логического 0.

Повторная запись данных в ячейку динамической памяти называется *регенерацией* данных. Регенерация ячейки динамической памяти производится при каждом обращении к ней. Однако при обычной интенсивности работы памяти микропроцессорной системы такая регенерация не гарантирует сохранности всех хранящихся в ней битов информации. Микропроцессор может, например, затратить время, превышающее несколько миллисекунд, на выполнение простого цикла, предназначенного для выработки временно́й задержки, и использовать при этом лишь несколько ячеек памяти. В течение всего этого временно́го промежутка все другие слова памяти не будут подвергаться регенерации.

По этой причине устройства динамической МОП-памяти снабжаются так называемой логической схемой регенерации. Эта схема автоматически осуществляет обращение к каждому столбцу памяти с интервалами в несколько десятых долей миллисекунды. Динамические запоминающие устройства построены таким образом, что само обращение к столбцу обеспечивает регенерацию информации во всех его ячейках. Работа логической схемы регенерации должна координироваться с другими действиями микропроцессора. Если, например, микропроцессор пытается обратиться к памяти в то время, когда в ней осуществляется регенерация, то схема регенерации должна отдать приоритет именно микропроцессору.

На рис. 11.5 показаны четыре варианта внешних связей, т.е. по существу это различные способы организации структуры интегральных схем статической МОП-памяти. На рис. 11.5, *а* изображено одно из первых статических ОЗУ емкостью 1К. Оно служит для хранения 1024 бит информации. Чтобы разместить 1024 8-разрядных слова, требуется восемь таких устройств. Каждое из них имеет 10 адресных линий, позволяющих осуществить выборку одного из 1024 бит ($2^{10} = 1024$). Наличие линии «Выбор кристалла» позволяет использовать 11-разрядный

адрес, причем с помощью 11-го разряда задается один из двух рядов, составленных из таких ОЗУ. Одиннадцатый разряд адреса поступает на входы «Выбор кристалла» интегральных схем, находящихся в первом ряду, непосредственно, а на входы «Выбор кристалла» интегральных схем второго ряда – через инвертор. Таким образом, когда производится выборка одного из рядов интегральных схем ОЗУ, выборка второго ряда не осуществляется.

Кроме линий адреса и выбора кристалла устройство снабжено линиями ввода и вывода данных, а также еще одной линией, по которой поступает сигнал, определяющий режим работы памяти – чтение или запись.

На рис. 11.5, *б* показана организация широко используемого статического ОЗУ емкостью 4096 бит. Это устройство также имеет линии адреса, данных, выбора кристалла и чтения/записи. Эта интегральная схема располагает объемом памяти, в четыре раза превышающим объем ОЗУ, показанного на рис. 11.5, *а*. Ее обычно называют статическим ОЗУ 4К.

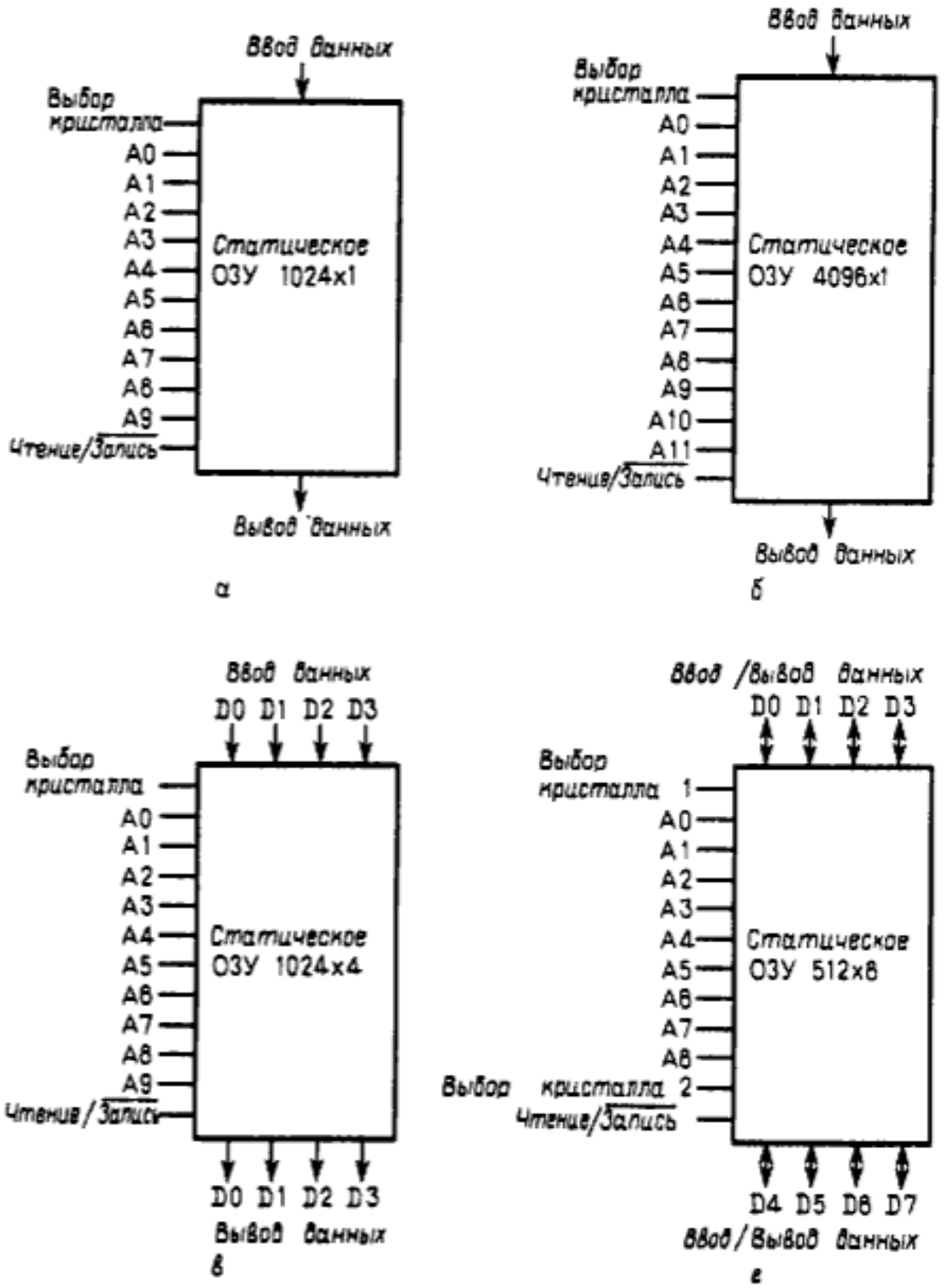


Рис. 11.5. Четыре способа организации статического ОЗУ: *а* – статическое ОЗУ емкостью 1К; *б* – статическое ОЗУ емкостью 4К; *в* – ОЗУ 1024 × 4 с полубайтовой организацией; *г* – ОЗУ 512 × 8 с байтовой организацией.

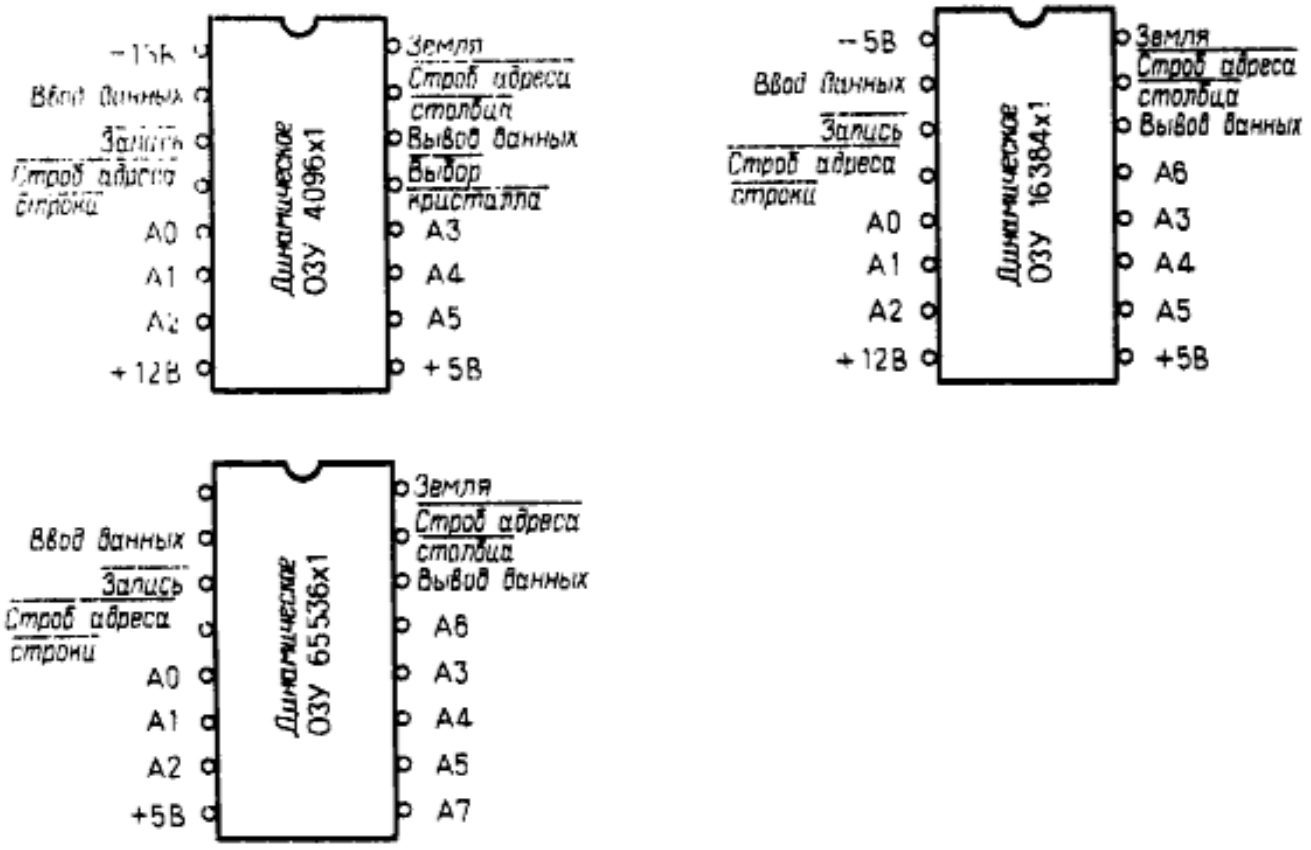


Рис. 11.6. Назначение выводов трех взаимозаменяемых динамических ОЗУ.

Рис. 11.5, *в* дает представление об организации другой интегральной схемы памяти объемом 4096 бит. Такая память предназначена для хранения 1024 4-разрядных слов. Это означает, что схема должна иметь четыре входные и четыре выходные линии данных. Каждый раз, когда адресуется память, производится либо считывание из нее полубайта, либо запись. Из двух таких схем можно сформировать память объемом 1К байт. Данный вид организации памяти весьма удобен для использования в системах небольшого объема.

На рис. 11.5, *г* показана организация памяти с байтовой ориентацией. Подобная 4096-битовая интегральная схема служит для хранения 512 байт. Даже одной схемы такого типа достаточно для применения в качестве ОЗУ небольшой микропроцессорной системы.

Организация большинства динамических ОЗУ предусматривает возможность их взаимозаменяемости. Это означает, что разводка связей печатной платы позволяет устанавливать в позиции, отведенные на плате для памяти, ОЗУ объемом 4К, 16К и 64К. За счет этого объем памяти микропроцессорной системы может быть выбран в соответствии с требованиями каждого пользователя. Часто бывает необходимо организовать управление платой памяти, на которой установлены разнотипные интегральные схемы памяти, а также изменить объем памяти микропроцессорной системы.

На рис. 11.6 показано назначение выводов трех интегральных схем памяти, представляющих собой также запоминающие устройства объемом 4К, 16К и 64К. Эти устройства размещаются в корпусах с 16 выводами, расположенными в два ряда (так называемых корпусах DIP). Для уменьшения числа внешних выводов при адресации в данном случае использован специальный прием. Обратите внимание, что над названиями некоторых входных сигналов, например «Выбор кристалла», имеется горизонтальная черта. Эта черта означает, что сигнал считается поданным на данный вход, если он имеет значение логического 0. Поясним сказанное на примере запоминающего устройства объемом 4К. На рис. 11.7 приведена схема, поясняющая используемый метод дешифрирования адреса. На первый взгляд эта схема выглядит в отношении внешних связей

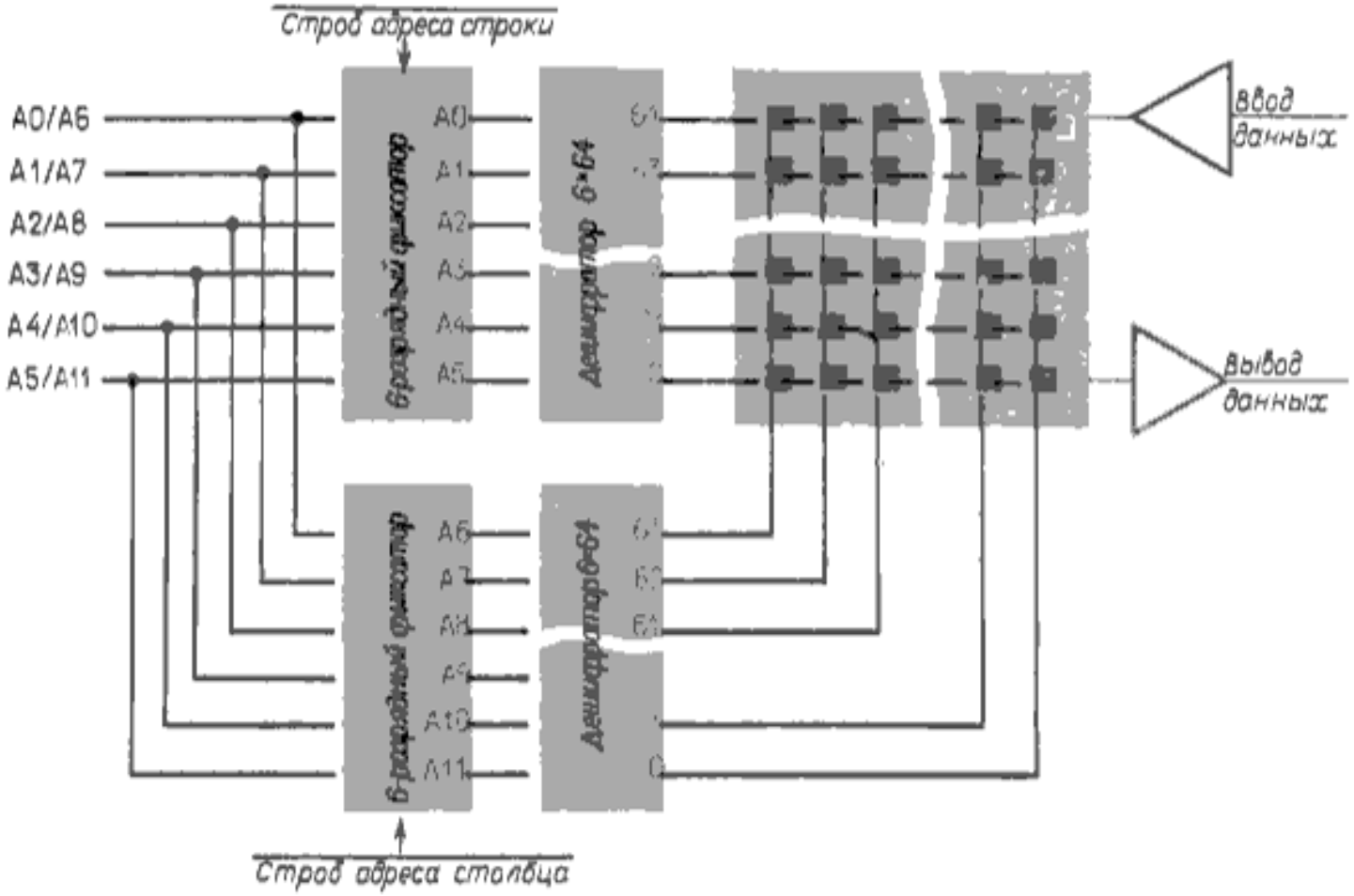


Рис. 11.7. Динамическое ОЗУ с мультиплексной передачей адресов строк и столбцов.

точно так же, как и схема памяти объемом 4096 бит, показанная на рис. 11.2. Для работы с памятью 4К используются дешифраторы адресов столбца и строки, каждый из которых имеет шесть входов и 64 выхода. На входе каждого дешифратора стоит 6-разрядный регистр-фиксатор. Для адресации ячейки в этом запоминающем устройстве необходимо затратить два временны́х цикла. Сначала на шесть адресных входов интегральной схемы поступают сигналы с шести линий младшей части адреса памяти A0–A5, после чего на вход «Строб адреса строки» подается сигнал, по которому эти шесть разрядов адреса записываются в регистр-фиксатор дешифратора строки.

Затем на шесть адресных входов интегральной схемы поступают шесть старших разрядов адреса. На этот раз сигнал подается на вход «Строб адреса столбца», в результате чего эти шесть старших разрядов адреса заносятся в фиксатор дешифратора столбца.

Теперь в регистрах-фиксаторах записаны все 12 бит адреса, что дает возможность выбрать одну из 4096 ячеек памяти. Описанный прием называется *адресацией с мультиплексированием.*

Как следует из рис. 11.6, все интегральные схемы ОЗУ имеют ряд одинаковых внешних выводов. Ввод данных в ОЗУ производится через контакт 2, а вывод – через контакт 14. Как нам известно, каждая ячейка ОЗУ содержит 1 бит информации, т.е. для запоминания 1 байт данных требуется восемь таких ячеек.

Каждая из рассматриваемых схем памяти имеет вход «Запись». На этот вход подается сигнал, когда необходимо записать данные в адресуемую ячейку памяти. Если же сигнал на этот вход не подан, схема пребывает в режиме чтения. В запоминающих устройствах предусмотрены также входы «Строб адреса строки» и «Строб адреса столбца», позволяющие использовать мультиплексирование адреса.

Динамическое ОЗУ объемом 4К имеет шесть адресных линий A0–A5, динамическое ОЗУ 16К – семь адресных линий A0–A6, а динамическое ОЗУ 64К – восемь адресных линий A0–A7. Нетрудно убедить-

ся, что за счет двукратного использования адресных линий во всех этих ОЗУ обеспечиваются необходимые диапазоны адресации, которые составляют соответственно

$$2^6 \times 2^6 = 2^{12} = 4096$$
$$2^7 \times 2^7 = 2^{14} = 16\,384$$
$$2^8 \times 2^8 = 2^{16} = 65\,536$$

Общими для указанных ОЗУ являются только адресные линии A0–A5. В ОЗУ 16К и 64К для адресации используется входной контакт, который в ОЗУ 4К служит для подачи входного сигнала «Выбор кристалла». По нему поступает разряд адреса A6. При необходимости реализовать функцию выбора кристалла следует применить внешние логические схемы.

Для функционирования ОЗУ 4К и 16К используются три питающих напряжения, подаваемые относительно общей линии земли: +12 В, +5 В и −5 В. Динамическое ОЗУ 64К работает только с одним источником питания (+5 В). Один из контактов, служащих в ОЗУ двух других типов для подачи напряжения питания, используется в ОЗУ 64К для подачи разряда адреса A7. Полупроводниковые схемы памяти с одним источником питания представляют собой одно из новейших достижений технологии. Их применение позволяет обеспечить высокую плотность компоновки ИС ОЗУ 64К. Как видно на схеме, ОЗУ 64К имеет один незадействованный внешний вывод. Если его использовать для подачи разряда адреса A8, то впоследствии в том же корпусе можно разместить динамическое ОЗУ объемом 262 144 бит (256К).

Задания для самопроверки

8. Укажите, с помощью какого из нижеперечисленных элементов или схем обеспечивается запоминание информации в статических ОЗУ:

а) конденсатор; б) триггер;

в) биполярная схема; г) МОП-схема?

9. Для выполнения каких из указанных ниже функций необходимы специальные схемы в динамических ОЗУ:

а) декодирование адреса строки; б) декодирование адреса столбца; в) регенерация; г) адресация?

10. Может ли память 4К иметь конфигурацию а) 4К × 1, б) 1К × 4, в) 512 × 8 или г) возможна любая из них?

11. Наличие линий выбора кристалла дает возможность использовать два кристалла с общими линиями а) питания, б) земли, в) адреса или г) инвертирования?

12. Используется ли а) напряжение питания −5 В, б) напряжение питания +12 В, в) корпус типа DIP и г) адресация с мультиплексированием в динамических ОЗУ большого объема (4К, 16К или 64К)?

13. Какая из указанных ниже операций выполняется при подаче сигнала на вход Запись:

а) запись данных в ОЗУ; б) адресация памяти; в) вывод данных на шину данных; г) фиксация данных в регистре?

14. Для работы памяти, в которой используется адресация с мультиплексированием, необходимы два стробирующих сигнала. Какие функции они выполняют? Зачем нужно так усложнять интегральные схемы памяти, чтобы организовать мультиплексирование адреса?

15. Объясните различие между способами запоминания данных в статических и динамических ОЗУ.

16. Зачем нужна логическая схема регенерации для динамических ОЗУ?

11.3. Пример построения платы памяти

В предыдущем разделе были рассмотрены интегральные схемы, предназначенные для построения статических и динамических запоминающих устройств на МОП-транзисторах. Каким образом используются эти схемы для реализации памяти микропроцессорных систем? Какие вспомогательные схемы должны быть размещены на плате памяти?

На рис. 11.8 приведена упрощенная структурная схема памяти небольшой микропроцессорной системы. Для построения памяти объемом 1К байт применены две интегральные схемы памяти объемом 512 8-разрядных слов каждая. В зависимости от структуры шины конкретной микропроцессорной системы, применяемых инте-

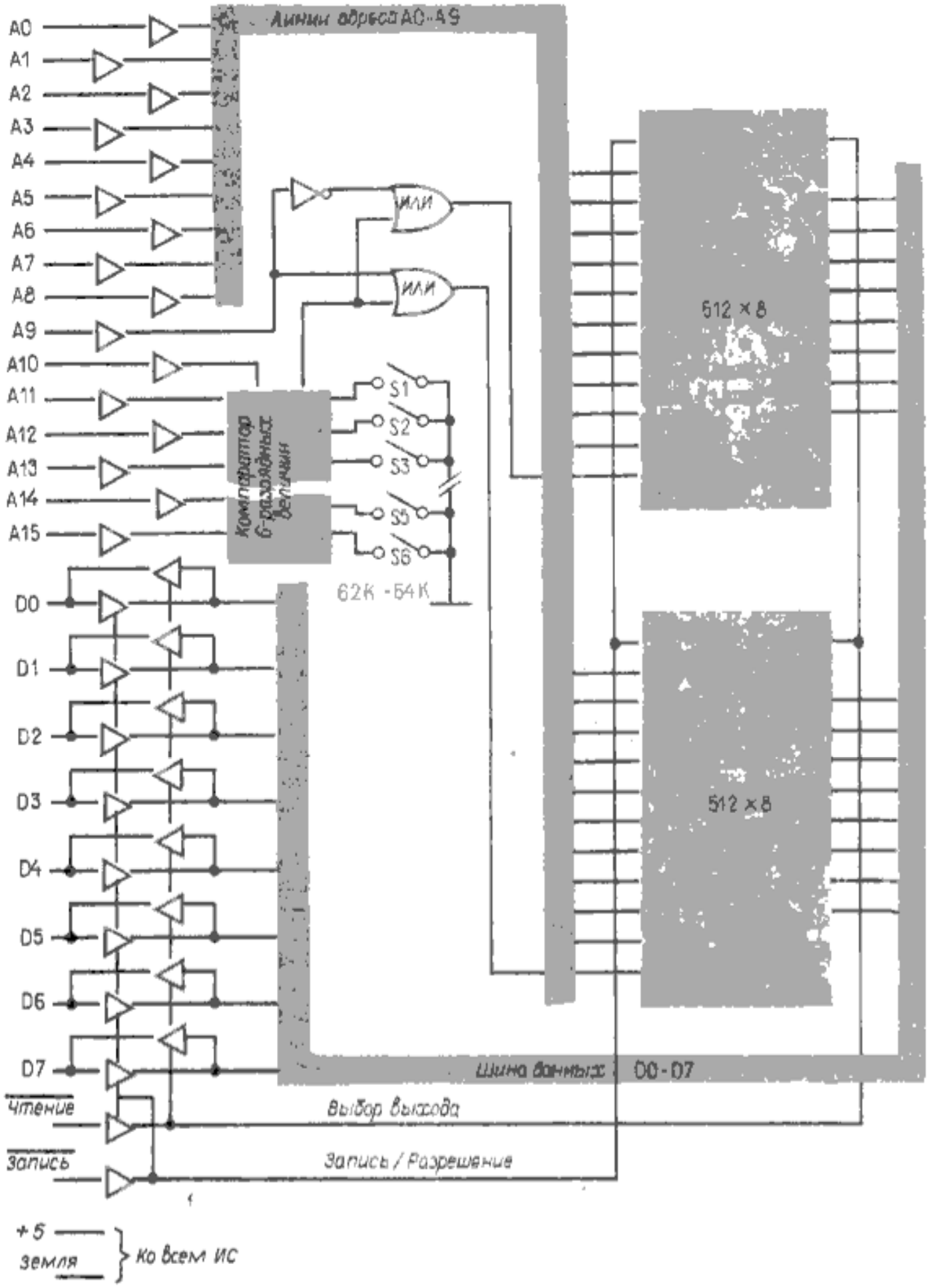


Рис. 11.8. Упрощенная структурная схема памяти объемом 1К байт. (Сигнал по линии «Запись/Разрешение» подается при необходимости записать в память байт данных. По линии «Выбор выхода» сигнал подается, когда нужно вывести байт данных из ИС на шину данных. Когда этот сигнал отсутствует, выходы D0 - D7 ИС памяти (линии ввода-вывода данных) пребывают в состоянии высокого импеданса.)

гральных схем памяти и имеющихся в наличии навесных ТТЛ-компонентов для построения реальной системы памяти могут потребоваться какие-то дополнительные интегральные схемы.

В левой части схемы показаны в упрощенном виде соединительные линии. Первые 16 линий, считая сверху, представляют собой адресные линии памяти A0–A15. Следующие восемь линий D0–D7 являются *двунаправленными* и составляют шину данных.

Для организации памяти необходимы еще четыре линии. По линии «Чтение» (Не чтение) подается сигнал, когда данные выводятся из памяти на шину данных микро-ЭВМ. По соседней линии «Запись» (Не запись) сигнал поступает, когда микро-ЭВМ осуществляет запись данных в память. Смысл обозначений «Чтение» и «Запись» состоит в том, что эти сигналы считаются активными, когда они имеют низкий уровень. Сделано так для упрощения организации системы электрических соединений. Одна из двух последних линий служит для подачи на плату памяти напряжения питания, другая – для заземления.

Как видно из схемы, все адресные сигналы подлежат буферированию. Это обычно осуществляется с помощью схем ТТЛ малого уровня интеграции. Буферы служат для развязки шины микро-ЭВМ и внутренних адресных линий интегральных схем памяти. В небольших схемах такие буферы могут не использоваться.

Адресные сигналы A0–A8 с буферов подаются непосредственно на интегральные схемы памяти.

Адресная линия A9 служит для выбора либо первой, либо второй интегральной схемы памяти. С помощью адресных линий A0–A8 можно адресовать любую ячейку в 512-байтовой области. Сигнал, поступающий по линии A9, позволяет выбрать первую или вторую из этих 512-байтовых областей.

Сигнал A9 проходит через схему ИЛИ, выходной сигнал которой подается непосредственно на первую интегральную схему памяти. Аналогичный сигнал на вторую интегральную схему памяти поступает с выхода другой схемы ИЛИ, на вход которой поступает инвертированное значение сигнала A9. Следовательно, когда один из кристаллов памяти оказывается выбранным, второй не выбран. Сигнал, поступающий на второй вход описанных схем ИЛИ, позволяет блокировать выбор обеих интегральных схем памяти независимо от значения сигнала A9.

Сигналы с адресных линий A10–A15 подаются на схему сравнения 6-разрядных значений. Шесть других сигналов, поступающих на этот компаратор, задаются с помощью шести переключателей. Последние обычно представляют собой микроминиатюрный блок тумблеров, конструктивно оформленный в корпусе типа DIP и установленный непосредственно на плате памяти. Набор сигналов, поступающих с адресных линий A10–A15, сравнивается с набором, заданным посредством тумблеров. Выходной сигнал схемы сравнения содержит информацию о том, имеет ли место обращение к данной 1К-байтовой области памяти.

Если, например, все адресные сигналы, подаваемые по линиям A10–A15, имеют нулевое значение и сигналы такого же значения подаются со всех переключателей (т.е. переключатели находятся в замкнутом положении), то со схемы сравнения на вышеупомянутые схемы ИЛИ поступает сигнал 0. Если же все сигналы на линиях A10–A15 имеют единичное значение, то и выходной сигнал схемы сравнения равен 1.

Предполагая, что все тумблеры установлены в состояние 0, мы указали тем самым, что рассматриваемая плата памяти должна реагировать на адреса, лежащие в диапазоне от 0 до 1К. Если же, например, тумблеры установлены в состояние 000010, то к данной плате будут относиться адреса в диапазоне от 2К до 3К. Рис. 11.9 иллюстрирует задание границ областей памяти, имеющих объем 1К байт.

Сигнал логического 0, поступающий со схемы сравнения, используется вместе с адресным сигналом, передаваемым по линии A9, в качестве разрешающего для схем ИЛИ. При наличии хотя бы одного сигнала, равного 1, на входе элемента ИЛИ на его выходе также имеет место единичный сигнал. Так как разрешающим значением сигнала для интегральной

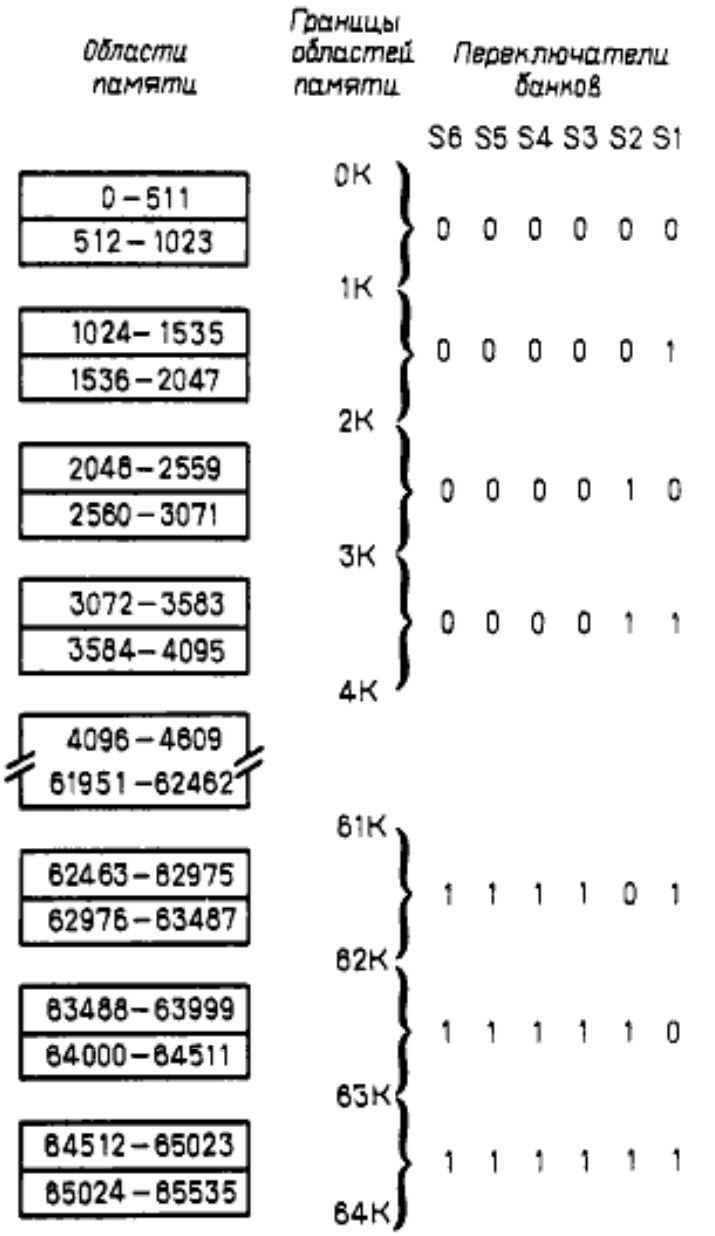


Рис. 11.9. Границы областей памяти объемом по 1К каждая в запоминающем устройстве, показанном на рис. 11.8. Наращивание памяти фрагментами по 1К от 0К до 63К осуществляется с помощью переключателей банков.

схемы памяти является 0, значение 1 оказывается для нее запрещающим. В данном случае сигнал разрешения не поступит ни на один из кристаллов, если адрес не находится в диапазоне 0К–1К.

Мы проследили процесс прохождения адреса на интегральные схемы памяти. Теперь рассмотрим, каким образом данные вводятся в память и выводятся из нее. Линии шины данных D0–D7 связаны с кристаллами памяти через шинные приемопередатчики (формирователи). Последние, представляющие собой интегральные схемы типа ТТЛ, выполняют следующие две функции.

Во-первых, эти схемы используются как буферы между линиями шины данных микро-ЭВМ и запоминающими устройствами, размещенными на плате памяти. Во-вторых, они служат для прохождения сигналов между платой памяти и шиной данных микро-ЭВМ. Таким образом, имеется возможность выводить данные из памяти на шину данных микро-ЭВМ и принимать данные в память с этой шины. Как можно видеть из схемы, группа буферов ввода и группа буферов вывода имеют общие управляющие сигналы. При подаче сигнала управления буферами ввода данные с шины данных микро-ЭВМ поступают на кристаллы памяти, а при подаче управляющего сигнала на буферы вывода данных выводятся на шину данных микро-ЭВМ из памяти.

Все сигналы управления буферами данных, а также сигналы для интегральных схем памяти («Запись/Разрешение», «Выбор выхода») вырабатываются в зависимости от значения сигналов «Чтение» и «Запись».

Рассмотрим пример выполнения операции чтения. На адресные линии памяти A0–A15 поступает некоторый адрес. Логическая схема обработки адреса выбирает ячейку в одной из двух интегральных схем памяти. Подается сигнал на линию управления памятью «Чтение». После буферирования этот сигнал по линии «Выбор выхода» поступает на соответствующий вход интегральных схем памяти. Из адресованной ячейки выбранного кристалла данные выводятся на внутреннюю шину данных платы памяти. Кроме того, под управлением сигнала «Чтение» внутренняя шина данных платы памяти соединяется с линиями D0–D7 шины данных микро-ЭВМ. Информация из адресованной ячейки платы памяти пересылается таким образом на шину данных микро-ЭВМ.

Итак, имеет место следующая последовательность действий:

1. Микропроцессор помещает на адресные линии памяти информацию, допустимую для адресации.

2. Устройство управления микропроцессора вырабатывает сигнал «Чтение».

3. Содержимое адресованной ячейки памяти поступает на внешние выводы кристалла памяти.

4. Выходные данные кристалла памяти подаются через буфер на шину данных микро-ЭВМ.

5. Микропроцессор принимает эти данные через порт шины данных и пересылает их по месту назначения.

Проследим теперь, как выполняется операция записи в память. Последовательность действий при этом такая же, изменяются лишь названия сигналов.

1. Микропроцессор помещает на адресные линии памяти информацию, допустимую для адресации.

2. Устройство управления микропроцессора вырабатывает сигнал «Запись».

3. Микропроцессор выводит данные через порт шины данных.

4. Информация с шины данных микро-ЭВМ поступает через буфер на входы данных микросхемы памяти.

5. Данные записываются в выбранную ячейку памяти.

В схеме, приведенной на рис. 11.8, использованы две интегральные схемы статического ОЗУ 4К. Для построения памяти такого же объема на основе двух интегральных схем динамического ОЗУ 4К потребовались бы дополнительные схемы. Они нужны для осуществления регенерации микросхем динамической памяти через каждые ~ 300 мкс. Применение динамической памяти увеличивает «накладные расходы» памяти. Под «накладными расходами» понимаются в данном случае дополнительные интегральные схемы, которые не предназначены собственно для хранения данных, но необходимы для обеспечения функционирования памяти. Так, для схемы, приведенной на рис. 11.8, к накладным расходам можно отнести интегральные схемы буферов данных и дешифраторов адреса. Они выполняют необходимые действия, но не выполняют функцию хранения данных.

Как правило, схема регенерации содержит такое же количество интегральных схем, как шинные буферы и логические схемы декодирования. В этом случае наличие схемы регенерации удваивает «накладные расходы». Можно теперь понять, почему динамические запоминающие устройства используются только для реализации памяти большого объема, а статические – для относительно небольшой памяти.

Рассмотрим плату динамической памяти более подробно. Если к пяти интегральным схемам шинных буферов и дешифраторов адреса добавить пять интегральных схем, предназначенных для выполнения регенерации, то общие затраты оборудования составят:

5 ИС – Дешифраторы адреса и шинные буферы (накладные расходы)
5 ИС – Схемы регенерации памяти (накладные расходы)
2 ИС – Динамические ОЗУ 512 × 8 (основные расходы)

---

12 ИС – Всего

Накладные расходы включают 10 интегральных схем, что составляет 83% аппаратных средств памяти.

Если, однако, задаться целью построить память объемом 64К на интегральных схемах ОЗУ 16К, то картина получится другая. Подсчитаем затраты оборудования в этом случае:

5 ИС – Дешифраторы адреса и шинные буферы (накладные расходы)
5 ИС – Схемы регенерации памяти (накладные расходы)
32 ИС – Динамические ОЗУ 16К × 1 (основные расходы)

---

42 ИС – Всего

Здесь накладные расходы составляют лишь 24% аппаратных средств памяти.

На этих простых примерах видно, что накладные расходы уменьшились с 83 до 24%. Конечно, имеются и другие предпосылки для широкого использования динамических ОЗУ. Не существует статических ОЗУ столь большого объема, как широко распространенные динамические ОЗУ; кроме того, динамические ОЗУ гораздо дешевле и имеют более высокое быстродействие.

Задания для самопроверки

17. Какие узлы схемы претерпели бы изменения, если бы для построения памяти,

показанной на рис. 11.8, были использованы восемь статических ОЗУ 4К × 1?

18. Как скажется увеличение объема памяти, проведенное в соответствии с п. 17, на накладных расходах аппаратных средств памяти? Объясните.

19. С помощью схемы, аналогичной схеме, приведенной на рис. 11.9, покажите, как будет происходить задание границ областей памяти и какого размера будут эти области применительно к памяти, определенной в п. 17.

20. Почему нецелесообразно использовать динамические запоминающие устройства для построения небольшой по объему памяти?

21. Часто одним из сигналов, поступающих по шине микро-ЭВМ, является «Допустимый адрес памяти». Для выработки каких специальных сигналов можно его использовать при организации функционирования платы памяти на динамических ОЗУ 16К?

11.4. Различные типы ПЗУ

Постоянное запоминающее устройство (ПЗУ) – очень важная составная часть любой микропроцессорной системы. Как следует из названия, ПЗУ – это такая память, информация в которой, будучи однажды записана, изменению не подлежит. Часто применение ПЗУ в микропроцессорной системе вызвано тем обстоятельством, что система всегда выполняет одну и ту же программу. Команды такой программы хранятся в ПЗУ. При использовании памяти этого типа не возникает проблемы энергозависимости, связанной с применением полупроводниковых ОЗУ. Любая микропроцессорная система должна включать какое-то ПЗУ, так как обязательно должна содержать встроенную программу, необходимую для загрузки в ОЗУ информации из внешней памяти большого объема, такой, как память на магнитной ленте или диске.

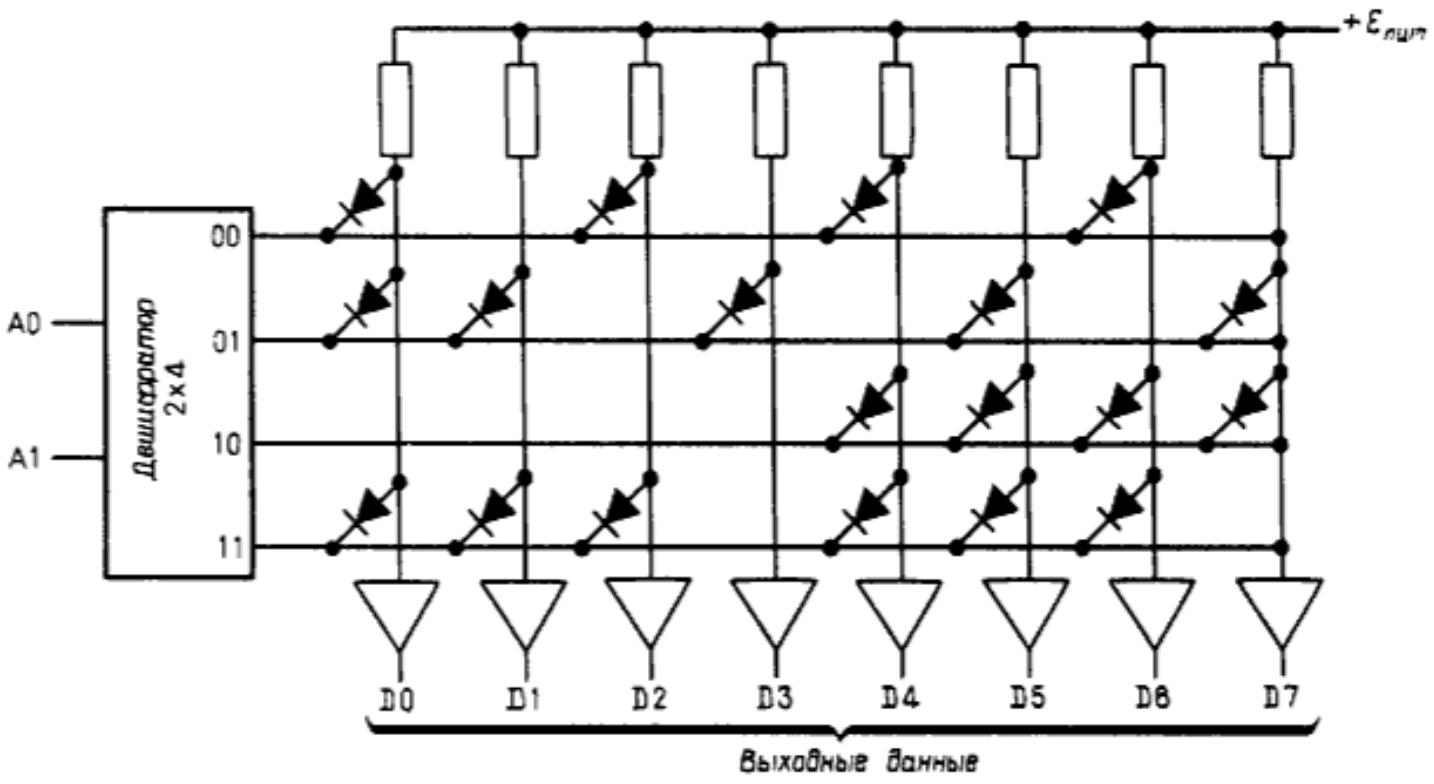
Существуют четыре типа ПЗУ различного назначения. Рассмотрим вначале простые ПЗУ. Говоря о запоминающем устройстве этого типа, имеют в виду память, в которую информация записана раз и навсегда в процессе изготовления полупроводниковых интегральных схем. Часто такие устройства называют *ПЗУ с масочным программированием*, так как набор битов, подлежащий запоминанию, фиксируется в ходе технологического процесса с использованием маскирующих фотошаблонов. Этот набор битов соответствует технической документации изготовителя. Постоянные запоминающие устройства применяются только в тех случаях, когда речь идет о массовом производстве, так как изготовление масок для интегральных схем частного применения обходится весьма недешево.

Далее перейдем к изучению *программируемого постоянного запоминающего устройства* (ППЗУ). Комбинация битов, вводимая в ППЗУ, может быть задана пользователем. Программирование ПЗУ – это однократно выполняемая операция, т. е. информация, когда-то записанная в ППЗУ, впоследствии изменена быть не может. Если необходимо сохранить в ППЗУ другую комбинацию битов, приходится вновь прибегнуть к программированию, но уже других микросхем ППЗУ.

Затем речь пойдет о *стираемом программируемом постоянном запоминающем устройстве* (СППЗУ), при работе с которым пользователь может запрограммировать его, затем стереть записанную информацию и вновь запрограммировать те же самые микросхемы. Будучи значительно более дорогими, чем ПЗУ и ППЗУ, СППЗУ дают пользователю возможность по мере надобности заменять содержащуюся в них информацию. Зачастую СППЗУ включают в комплект поставки микропроцессорных систем, функции которых окончательно определяются пользователем исходя из опыта применения.

Наконец, предметом обсуждения явится *электрически изменяемое постоянное запоминающее устройство* (ЭИПЗУ), программирование и изменение содержимого которого осуществляются с помощью электрических средств. В отличие от СППЗУ для стирания информации, хранимой в ЭИПЗУ, не требуется специальных внешних устройств.

На рис. 11.10 приведена схема простого ПЗУ. Для его реализации достаточно использовать дешифратор на ТТЛ-схемах и несколько диодов. Так как это ПЗУ содержит четыре 8-разрядных слова, его общий



Содержимое ПЗУ

Слово	*Двоичное представление*								*Шестнадцатеричное представление*
	D0	D1	D2	D3	D4	D5	D6	D7	
00	0	1	0	1	0	1	0	1	55
01	0	0	1	0	1	0	1	0	2A
10	1	1	1	1	0	0	0	0	F0
11	0	0	0	1	0	0	0	1	11

Рис. 11.10. Простое ПЗУ объемом четыре слова (4 байт).

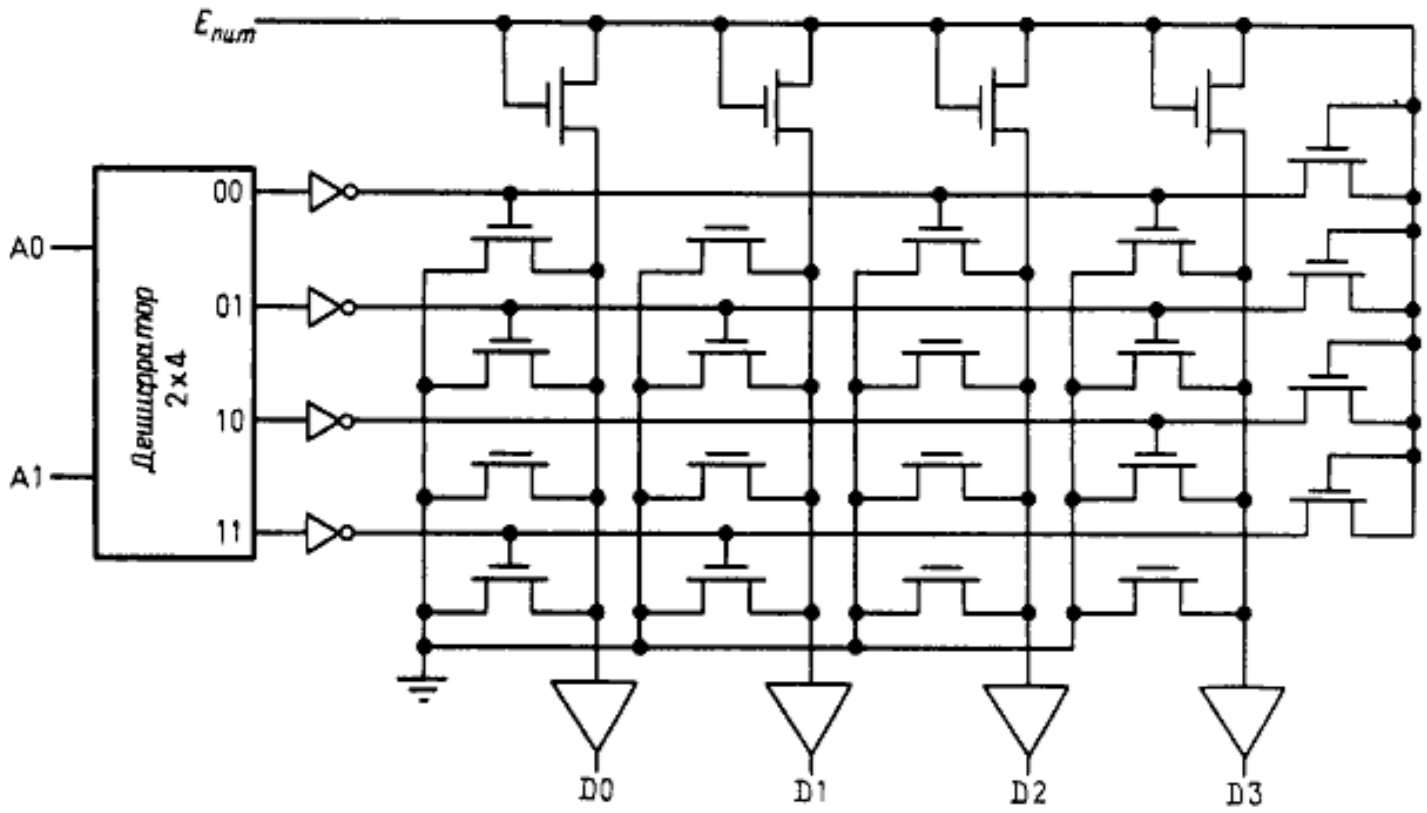
объем составляет 32 бит. Диоды установлены в тех местах ПЗУ, где должны храниться биты, имеющие значение логического 0.

Дешифратор 2 × 4 подает сигнал 0 на выбранную строку. На тех выходных вертикальных линиях, в пересечении которых с выбранной горизонтальной линией установлены соединительные диоды, формируются сигналы логического 0.

В настоящее время бóльшая часть ПЗУ изготавливается с использованием МОП-технологии. Биполярные ПЗУ изготавливаются в небольшом количестве для применения в тех случаях, когда необходимо очень высокое быстродействие. В ПЗУ на базе МОП-схем роль диода выполняет МОП-транзистор. Очень простое ПЗУ такого типа, предназначенное для хранения четырех слов, показано на рис. 11.11. Так как длина каждого слова 4 разряда, общий объем ПЗУ составляет 16 бит.

Как можно видеть на схеме, к каждой вертикальной линии (столбцу) подключены четыре МОП-транзистора. Через нагрузочный МОП-транзистор на все вертикальные линии подается напряжение питания. Затворы некоторых МОП-транзисторов подключены к горизонтальным линиям (строкам). Когда строка выбрана, на нее поступает высокий уровень напряжения. Если затвор МОП-транзистора подсоединен к выбранной строке, то транзистор открывается, и сигнал на соответствующей вертикальной линии приобретает значение логического 0 (его уровень приближается к потенциалу «земли»). При этом на выходе данного разряда ПЗУ формируется сигнал 0. Если затвор МОП-транзистора к выбранной строке не подсоединен, то транзистор не открывается, и на выходе данного разряда ПЗУ сигнал имеет значение, равное 1.

ПЗУ изготавливаются посредством масочного программирования. Как известно,



Содержимое ПЗУ

Слово	*Двоичное представление*				*Шестнадцатеричное представление*
	D0	D1	D2	D3	
00	0	1	0	0	4
10	0	0	1	0	2
01	1	1	1	0	E
11	0	0	1	1	3

Рис. 11.11. Простое ПЗУ объемом четыре слова (длиной по полубайту каждое).

активные компоненты микросхем соединяются между собой путем нанесения на подложку очень тонких металлических проводников, которые, подобно проводникам печатной платы, образуют внутренние соединения интегральной схемы. Конфигурацию последнего слоя металлизации ПЗУ, наносимого с помощью масочного программирования, определяет пользователь. Именно в этом слое формируются связи затворов некоторых МОП-транзисторов со строками. Затворы транзисторов, соответствующих разрядам, которые должны иметь значение 1, не соединяются со строками.

ПЗУ с масочным программированием изготавливаются при серийном производстве. Как правило, они не используются, пока изготовитель не найдет целесообразным организовать производство нескольких тысяч идентичных микросхем ПЗУ. Такие ПЗУ применяются в игрушках с микропроцессорным управлением, телевизионных играх, ЭВМ бытового назначения и в других аналогичных изделиях.

Рис. 11.12 иллюстрирует один из способов изготовления ПЗУ, программируемых пользователем. Такое ППЗУ носит название *ПЗУ с пережигаемыми перемычками*. Это устройство представляет собой ПЗУ на диодах, аналогичное показанному на рис. 11.10. При покупке такое ППЗУ содержит диоды и пережигаемые перемычки в позициях всех разрядов. Программирование такого ППЗУ заключается в том, что на него подаются адреса слов и разрушаются перемычки в тех местах, где они не нужны. После того как это сделано, содержимое ППЗУ изменению не подлежит. Если при программировании имела место ошибка, то единственный способ ее исправить – выбросить негодную микросхему и произвести программирова-

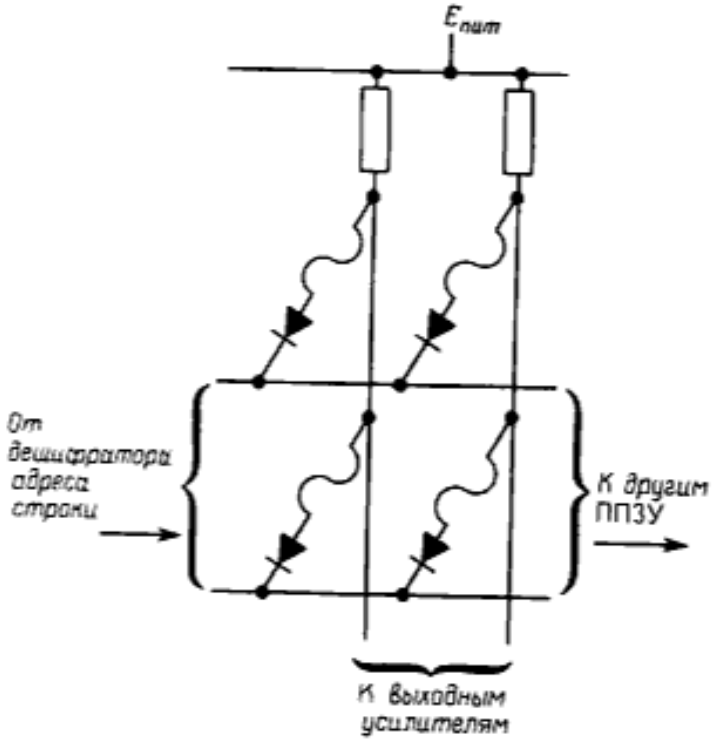


Рис. 11.12. ППЗУ с пережигаемыми перемычками.

ние другой. Так как программирование подобных ППЗУ с пережигаемыми перемычками может быть легко осуществлено за пределами предприятия-изготовителя, их часто называют *ПЗУ, программируемыми в полевых условиях.*

Построение ПЗУ с возможностью стирания информации в значительной степени аналогично организации ПЗУ с масочным программированием. Его основу составляет такая же матрица МОП-транзисторов, какая показана на рис. 11.11. Однако между СППЗУ и ПЗУ имеется ряд различий. Во-первых, затворы всех транзисторов в СППЗУ соединены с горизонтальными строками. Во-вторых, конструкция транзистора предусматривает возможность подачи на его затвор высокого напряжения. Если такое напряжение подано, то транзистор сохраняет высокое сопротивление. В-третьих, транзистор может быть выведен из этого состояния путем воздействия на него интенсивного ультрафиолетового излучения.

Программирование СППЗУ может быть выполнено в лабораторных условиях подачей импульсов высокого напряжения. Будучи запрограммировано, СППЗУ сохраняет информацию до тех пор, пока не будет осуществлена засветка кристалла интенсивным ультрафиолетовым излучением через кварцевое окошко, имеющееся на корпусе микросхемы. В результате облучения содержимое СППЗУ стирается.

Широкое распространение получает сейчас четвертый тип ПЗУ – электрически изменяемое ПЗУ. Организация ЭИПЗУ в значительной степени схожа с организацией СППЗУ. Однако ЭИПЗУ присуща одна особенность. Содержимое ЭИПЗУ может быть изменено путем подачи особых электрических сигналов в течение определенного времени. Обычно длительность сигналов записи должна составлять несколько миллисекунд, в результате чего процесс записи протекает значительно медленнее процесса чтения информации. ЭИПЗУ не обеспечивает столь долговременного хранения информации, как СППЗУ. Однако ЭИПЗУ может быть запрограммировано пользователем и относится к классу энергонезависимых запоминающих устройств.

ЭИПЗУ находят применение в таких приборах, как селекторы каналов в телевизионных установках, а также используются для временного запоминания кадров в видеотерминалах.

Итак, мы рассмотрели четыре типа ПЗУ. Запоминающие устройства с масочным программированием обеспечивают постоянное хранение информации. Они, кроме того, являются самыми дешевыми ПЗУ и позволяют запоминать наибольшее число битов. ППЗУ могут быть запрограммированы в «полевых условиях», но только однократно. Их стоимость лежит между стоимостью СППЗУ и ПЗУ с масочным программированием. СППЗУ могут быть запрограммированы в полевых условиях и, кроме того, допускают многократное обновление запоминаемой информации. Плотность записи информации в них меньше, чем в ПЗУ первых двух типов. Информация, запоминаемая в ЭИПЗУ, может быть обновлена с помощью электрических сигналов. Этот тип ПЗУ имеет наибольшую стоимость и характеризуется наименьшей плотностью размещения информации.

На рис. 11.13 показано назначение внешних выводов корпусов двух широко применяемых микросхем СППЗУ: 2716 и 2732. Микросхема 2716 – это ППЗУ с возможностью стирания ультрафиолетовыми лучами, имеющее объем 16К бит (2К × 8); микросхема 2732 – аналогичное ППЗУ, объем

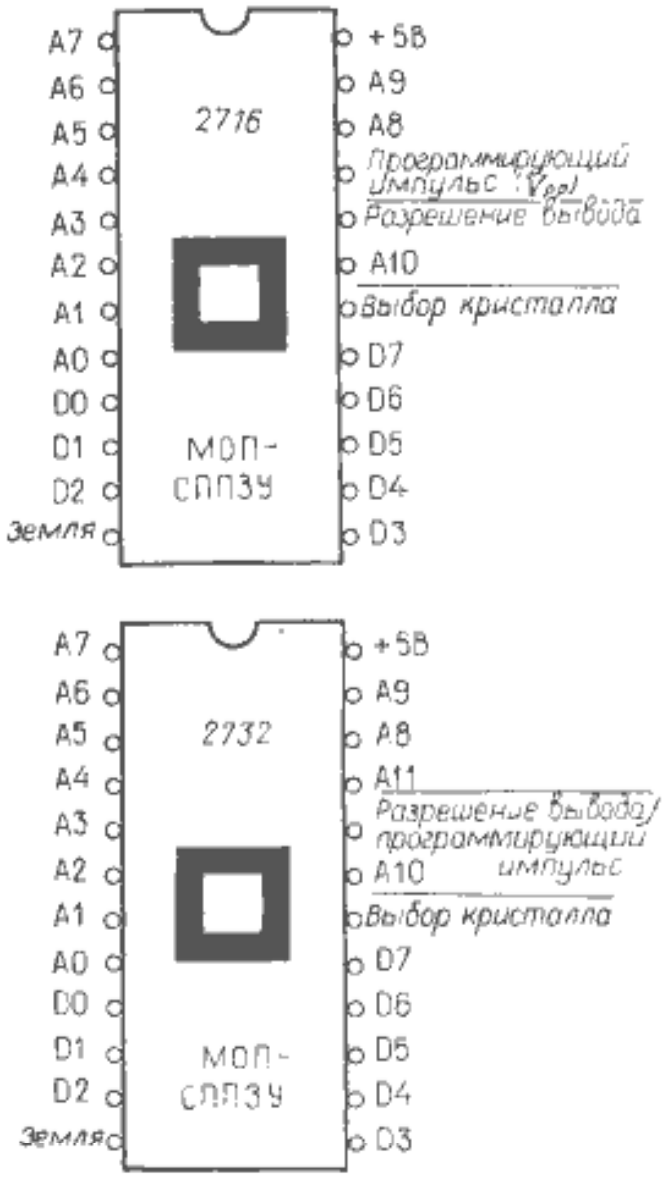


Рис. 11.13. СППЗУ типа 2716 и 2732. (В процессе нормальной работы программирующий вход не используется. Он предусмотрен для работы со специальным программатором СППЗУ.)

которого составляет 32К бит (4К × 8). Обе эти интегральные схемы памяти имеют байтовую организацию. Внешний вывод 20 микросхемы типа 2732 используется для передачи двух управляющих сигналов, тем самым высвобождается вывод 21, который отводится в микросхеме этого типа для подключения адресной линии A11. Использование для двух сигналов еще одного внешнего вывода позволяет разместить на кристалле микросхемы ППЗУ с возможностью стирания ультрафиолетовым излучением объемом 64К бит. СППЗУ, имеющие объем 8К, 16К, 32К и 64К бит, как и динамическое ОЗУ с объемом 4К, 16К и 64К бит, построены таким образом, что любое из них может быть изъято с платы и заменено запоминающим устройством, имеющим более высокую плотность информации.

В качестве адресных линий здесь используются линии A0–A10 (A11), а данные выводятся на линии D0–D7. В обеих микросхемах, показанных на рис. 11.13, используется по одному внешнему выводу для подачи постоянного напряжения + 5 В, заземления и сигнала «Выбор кристалла». По сигналу «Разрешение вывода» данные подаются на внешние выводы D0–D7. При программировании запоминающего устройства на контакты A0–A10 (A11) поступает адресная информация, на контакты D0–D7 – записываемые данные, а на внешний вывод U_{pp} – импульс напряжения 25 В длительностью 50 мс. Для стирания информации микросхема СППЗУ облучается через имеющееся на ней кварцевое окошко светом специальной ультрафиолетовой лампы в течение 15–20 мин. Количество вырабатываемой при этом энергии ультрафиолетового облучения примерно такое же, какое было бы получено в течение трех лет от люминесцентной лампы или за неделю облучения прямыми солнечными лучами. На некоторых интегральных схемах СППЗУ кварцевое окошко закрывается специальной крышкой, чтобы предотвратить случайное стирание информации.

Существование четырех типов ПЗУ, значительно различающихся по плотности записи информации, создает широкие возможности для выбора ПЗУ. Имеются ПЗУ с масочным программированием, которые могут быть установлены в контактные колодки, используемые для СППЗУ. Это очень удобно, так как позволяет изготовителям приборов применять СППЗУ при разработке и на первых этапах производства. Ошибки, допущенные при разработке программ, могут быть исправлены путем перезаписи информации в СППЗУ. Впоследствии, после отработки программного обеспечения микросхемы, СППЗУ могут быть заменены более дешевыми микросхемами ПЗУ.

Кроме того, промышленность изготовляет ПЗУ с масочным программированием, ориентированные на некоторые типичные применения. В такие ПЗУ записываются определенные стандартные таблицы.

Имеются, например, ПЗУ, предназна-

ченные для использования в терминалах с дисплеями. Эти ПЗУ содержат изображения всех буквенно-цифровых символов в виде матриц точечных изображений 5 × 7.

Задания для самопроверки

22. Какие из перечисленных ниже типов ПЗУ наиболее дорогостоящие:

а) ПЗУ с масочным программированием;

б) ПЗУ, допускающие программирование в полевых условиях;

в) программируемые ПЗУ, предусматривающие возможность стирания информации;

г) ПЗУ с возможностью изменения информации электрическим путем?

23. В ПЗУ какого типа имеются пережигаемые перемычки:

а) ПЗУ с масочным программированием;

б) ПЗУ, допускающие программирование в полевых условиях;

в) программируемые ПЗУ с возможностью стирания информации;

г) электрически изменяемые ПЗУ?

24. Хотя имеется принципиальная возможность заказать ПЗУ, ППЗУ, СППЗУ и ЭИПЗУ с уже записанной в них в соответствии с потребностями пользователя информацией, только в ПЗУ одного типа эта запись производится уже в ходе технологического процесса изготовления микросхем памяти. Укажите этот тип ПЗУ из нижеперечисленных устройств:

а) ПЗУ с масочным программированием;

б) ПЗУ, допускающие программирование в полевых условиях;

в) программируемые ПЗУ с возможностью стирания информации;

г) электрически изменяемые ПЗУ.

25. Интегральная схема памяти 2708 относится к тому же классу программируемых ПЗУ со стиранием информации посредством ультрафиолетового излучения, что и микросхемы 2716 и 2732; объем хранимой в ней информации составляет 8К. Укажите, какое число адресных линий [а) 8, б) 9, в) 10, г) 11 или д) 12] должна иметь указанная интегральная схема.

26. Почему СППЗУ являются наиболее широко используемым типом постоянных запоминающих устройств при разработке микропроцессорных систем?

27. Почему микро-ЭВМ должна содержать некоторый объем постоянной памяти?

28. Почему построение памяти микро-ЭВМ полностью на интегральных схемах СППЗУ представляется маловероятным?

11.5. Прямой доступ к памяти

Все пересылки данных, имеющие место при их вводе и выводе, которые мы рассматривали до сих пор, происходили в ходе выполнения программ. При вводе данные поступали из регистра входных данных в аккумулятор, а затем направлялись в память. Программно управляемый вывод данных подразумевает их пересылку из памяти в аккумулятор, а потом из аккумулятора в выходной регистр.

К сожалению, программно управляемая пересылка данных – это весьма медленный процесс. При пересылке больших массивов данных это обстоятельство обычно вызывает определенные проблемы.

Предположим, что нужно осуществить пересылку 16К байт данных с магнитной ленты. Каждый сантиметр магнитной ленты содержит 311 байт данных. Механизм перемотки перемещает ленту со скоростью 316,25 см/с. Для размещения 16К байт данных используется 51,9 см ленты. Это означает, что пересылка 16 384 байт данных занимает 0,16384 с. Другими словами, данные пересылаются со скоростью 100 000 байт/с.

На рис. 11.14 приведена короткая программа, которая может быть использована для пересылки данных с магнитной ленты под управлением программы. Можно подсчитать количество циклов микропроцессора, затрачиваемых на ее выполнение, и определить время, необходимое для решения задачи. Блок данных, имеющий объем 16К байт, будет записываться в память, начиная с области 0400 и заканчивая областью 4400.

Подпрограмма программно управляемой пересылки данных размещается в памяти, начиная с области DATAIN. В подпрограмме использованы символические адреса, чтобы можно было использовать ее со-

Символический адрес	Код операции		Операнд	Комментарий
DATAIN	LDA	D	44H	;Загрузка в D конечного значения счетчика
	LRP	B	0004H	;Загрузка в BC указателя начала файла
INAGN	IN		01H	;Ввод данных в аккумулятор
	STI	A		;Запись содержимого аккумулятора в файл
	IRP	B		;Положительное приращение указателя файла
	MOV	A,B		;Пересылка содержимого B в A
	CMP	D		;В файл записано 16К байт?
	JNZ		INAGN	;Нет, продолжить ввод данных
				;Да, подпрограмма выполнена; возврат из подпрограммы

Рис. 11.14. Подпрограмма загрузки 16 384 байт данных в файл памяти. (Программа приведена в виде, предполагающем использование ассемблера. Для ассемблирования программы следует использовать команду ORG, операнд которой задает начальный адрес. Буква H в записи операндов обозначает их представление в шестнадцатеричном формате.)

вместно с любыми другими необходимыми программами. При ассемблировании подпрограммы символическое имя DATAIN будет преобразовано в абсолютный адрес. С помощью двух первых команд подпрограммы в регистр D загружается число 44, а в регистровую пару BC – указатель начала файла (0400). После записи в память каждого байта содержимое регистра D сравнивается с содержимым регистра B. Когда содержимое регистра B в результате положительных приращений становится равным 44, происходит выход из подпрограммы. Таким образом, 16 384-й байт будет загружен в область памяти 4400.

Выполнение подпрограммы начинается с символического адреса INAGN. По команде ВВОД данные пересылаются из порта ввода-вывода 01 в аккумулятор. По команде ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ КОСВЕННАЯ содержимое аккумулятора загружается в область памяти 0400. Затем с помощью команды IRP B осуществляется положительное приращение регистровой пары BC.

При выполнении команды MOV A,B содержимое регистра B загружается в аккумулятор. Можно теперь сравнить находящееся в аккумуляторе содержимое регистра B с величиной, записанной ранее в регистр D. Делается это с помощью команды CMP D.

С помощью этой команды выясняется, не стало ли содержимое регистровой пары BC в результате последнего выполнения команды положительного приращения указывать на область памяти 4400. Если это так, то выполнение задачи на этом завершается; если же имеет место другая ситуация, то необходимо вернуться к начальной части программы. Значение разряда нулевого результата регистра состояния проверяется с помощью команды JNZ. Если этот разряд не установлен в состояние 1, выполнение задачи не заканчивается, и в программе происходит возврат к команде INAGN. При завершении выполнения задачи разряд нулевого результата в регистре состояния устанавливается в 1. Поэтому команда перехода в 16 384-й раз не выполняется.

Рис. 11.15 иллюстрирует временны́е характеристики данной программы. Для организации файла бо́льшая часть команд программы должна быть выполнена 16 384 раза. Если предположить, что длительность микроцикла составляет 1 мкс, решение задачи займет в общей сложности 294 915 мкс, т. е. ~0,3 с. Таким образом, решение задачи, которое должно было продолжаться 0,16 с, занимает почти в два раза большее время вследствие того, что была использована программно управляемая пересылка данных.

Команда	Количество раз использования команды	Длительность выполнения одной команды при микроцикле, равном 1 мкс, мкс	Общая длительность выполнения команды, мкс
LDA D	1	3	3
LRP B	1	5	5
IN	16384	5	81920
STI A	16384	2	32768
IRP B	16384	2	32768
MOV A,B	16384	2	32768
CMP D	16384	2	32768
JNZ	16383	5	81915
			294915

Рис. 11.15. Временны́е характеристики программы, листинг которой приведен на рис. 11.14. (Команда JNZ при последнем цикле программы не выполняется; эта команда используется 16383 раза.)

Даже если применить процессор, длительность микроцикла которого составит 0,5 мкс, он едва будет поспевать за скоростью поступления данных с магнитной ленты.

Цель рассмотрения данного примера заключалась в том, чтобы показать необходимость наличия *прямого доступа к памяти* (*ПДП*). Прямой доступ к памяти – это один из способов организации быстрой пересылки данных при обмене информацией с памятью. Известно, например, что время доступа к памяти с относительно хорошими характеристиками памяти микропроцессорной системы лежит в диапазоне 300–500 нс. При столь малом времени доступа к памяти микропроцессор должен быть способен выполнять не менее 2 млн. пересылок в секунду. При этом мы только что убедились, что при программном управлении число пересылок в секунду не может выйти за пределы выше $\sim 25\,000$.

В большинстве случаев ПДП реализуется путем использования специального контроллера ПДП. На время выполнения ПДП микропроцессор должен быть лишен возможности обращения к памяти. Для того чтобы начать реализацию режима ПДП, микропроцессор загружает в контроллер ПДП содержимое некоторого внешнего регистра, представляющее собой начальный адрес файла данных. Кроме того, в другой регистр (счетчик пересылок), находящийся в контроллере ПДП, загружается количество байтов, подлежащее пересылке. Затем микропроцессор переходит в режим запрещения подачи адресов и данных по шинам и передает управление памятью контроллеру ПДП. Контроллер ПДП последовательно подает на адресную шину памяти микропроцессорной системы адреса и вырабатывает сигналы управления чтением и записью. При пересылке каждого байта осуществляется отрицательное приращение содержимого регистра-счетчика. Когда в результате отрицательных приращений содержимое счетчика становится равным 0, во внешнее устройство выдается сообщение, что пересылка данных завершена. Будучи узко специализированным устройством, контроллер ПДП выполняет все эти действия очень быстро. На сегодняшний день время ПДП-пересылок практически приближается к времени цикла памяти. Закончив пересылку данных в память или из нее, контроллер ПДП снова передает управление микропроцессору.

В настоящее время промышленность выпускает контроллеры ПДП в интегральном исполнении. Хотя эти контроллеры и не обладают таким быстродействием, как устройства на дискретных ТТЛ-компонен-

тах, значение этого параметра обычно сопоставимо с временем цикла памяти. Применение интегральных контроллеров ПДП значительно проще, чем устройств на дискретных ТТЛ-компонентах.

Задания для самопроверки

29. Предположим, что характеристики контроллера ПДП и памяти позволяют осуществлять 1 млн. пересылок в секунду. Пересылка ведется с магнитной ленты с плотностью записи 316,2 байт/см, перемещающейся со скоростью 189,75 см/с. Параметры какого из устройств – контроллера ПДП или устройства хранения информации на магнитной ленте – накладывают при этом ограничения на скорость пересылки данных?

30. Чем будет ограничена скорость пересылки данных, если в ситуации, описанной в п. 29, применить устройство хранения информации на магнитной ленте с плотностью хранения информации 632,4 байт/см и скоростью перемотки 316,25 см/с?

31. При программно управляемой пересылке данных для передачи каждого байта данных требуется только одна пересылочная операция. Почему в этом случае программно управляемая пересылка выполняется медленнее, чем пересылка с прямым доступом к памяти?

32. Имеется контроллер ПДП, управляющий работой четырех каналов прямого доступа к памяти. В состав контроллера входит восемь 16-разрядных регистров. Зачем они нужны?

Упражнения

11.1. Какое определение [а) динамическое, б) статическое, в) ПЗУ, г) энергозависимое] нельзя использовать для полупроводникового ОЗУ с произвольным доступом?

11.2. Для хранения каких видов информации [а) программы начального запуска, б) главной программы или в) команд] обычно используется энергонезависимая память?

11.3. Какое из приведенных ниже определений относится к полупроводниковой оперативной памяти:

а) энергозависимая,
б) статическая,
в) динамическая,
г) энергонезависимая?

11.4. Является время доступа к памяти промежутком времени между а) циклами регенерации, б) поступлением адреса и выводом данных, в) следующими друг за другом обращениями?

11.5. Какие из указанных ниже устройств должны входить в состав микропроцессорной системы:

а) ПЗУ,
б) оперативная память,
в) порты ввода-вывода?

11.6. Какой вид памяти [а) динамическая, б) биполярная, в) МОП или г) статическая] является наиболее широко распространенным?

11.7. Какие устройства представляют собой устройства на ПЗС и ПМД: а) с последовательным доступом, б) с произвольным доступом, в) типа ПЗУ, г) устаревших типов?

11.8. В каких элементах или устройствах [а) ПЗУ, б) ОЗУ, в) триггерах, г) конденсаторах] производится хранение данных в случае использования статических ОЗУ?

11.9. Какое из действий [а) регенерация, б) произвольный доступ, в) последовательный доступ] осуществляется при использовании динамической памяти?

11.10. Каков объем [а) 1024 бит, б) 2048 бит, в) 4096 бит, г) 8192 бит] статических ОЗУ 4К × 1 и 1К × 4?

11.11. В каких элементах или устройствах [а) ПЗУ, б) ОЗУ, в) триггерах, г) конденсаторах] производится хранение данных в случае использования динамической памяти?

11.12. При одинаковом объеме микросхем чем характеризуются устройства динамической памяти по сравнению с устройствами статической памяти:

а) отсутствием накладных расходов,
б) бóльшими накладными расходами,
в) меньшими накладными расходами,
г) теми же накладными расходами?

11.13. Какой из нижеприведенных видов адресации используется в динамических ОЗУ большого объема:

а) с мультиплексированием, б) по строкам и столбцам, в) двоичная?

11.14. ПЗУ является а) энергозависимым, б) энергонезависимым или в) динамическим запоминающим устройством?

11.15. К какому типу нижеперечисленных устройств относится СППЗУ:

а) с масочным программированием;
б) с пережигаемыми перемычками, допускающим программирование в полевых условиях;
в) с произвольным доступом, допускающим чтение и запись информации;

г) с произвольным доступом, допускающим только чтение информации?

11.16. К какому типу указанных ниже устройств относится ПЗУ:

а) с масочным программированием;

б) с пережигаемыми перемычками, допускающим программирование в полевых условиях;

в) с произвольным доступом, допускающим чтение и запись информации;

г) с произвольным доступом, допускающим только чтение информации.

11.17. Как сокращенно обозначаются программируемые постоянные запоминающие устройства с пережигаемыми перемычками: а) ОЗУ, б) ПЗУ, в) ППЗУ или г) СППЗУ?

11.18. Для какой из указанных целей используется прямой доступ к памяти:

а) пересылка данных в память через аккумулятор;

б) пересылка данных из памяти через аккумулятор;

в) прекращение работы микропроцессора на время пересылки данных;

г) достижение высокой скорости пересылки данных?

11.19. Объясните назначение входного сигнала «Выбор кристалла» в микросхемах ОЗУ и ПЗУ.

11.20. Каким образом можно обеспечить хотя бы в некоторой степени энергонезависимость полупроводниковых ОЗУ? Почему эти меры имеют лишь ограниченные возможности?

11.21. В контроллере ПДП имеются два регистра, содержимое которых загружается микропроцессором перед началом работы в режиме ПДП. Какие функции выполняют эти регистры? Почему они должны быть 16-разрядными?

11.22. Почему на время пересылки ПДП микропроцессор переходит в режим запрета передачи по шинам адреса и данных?

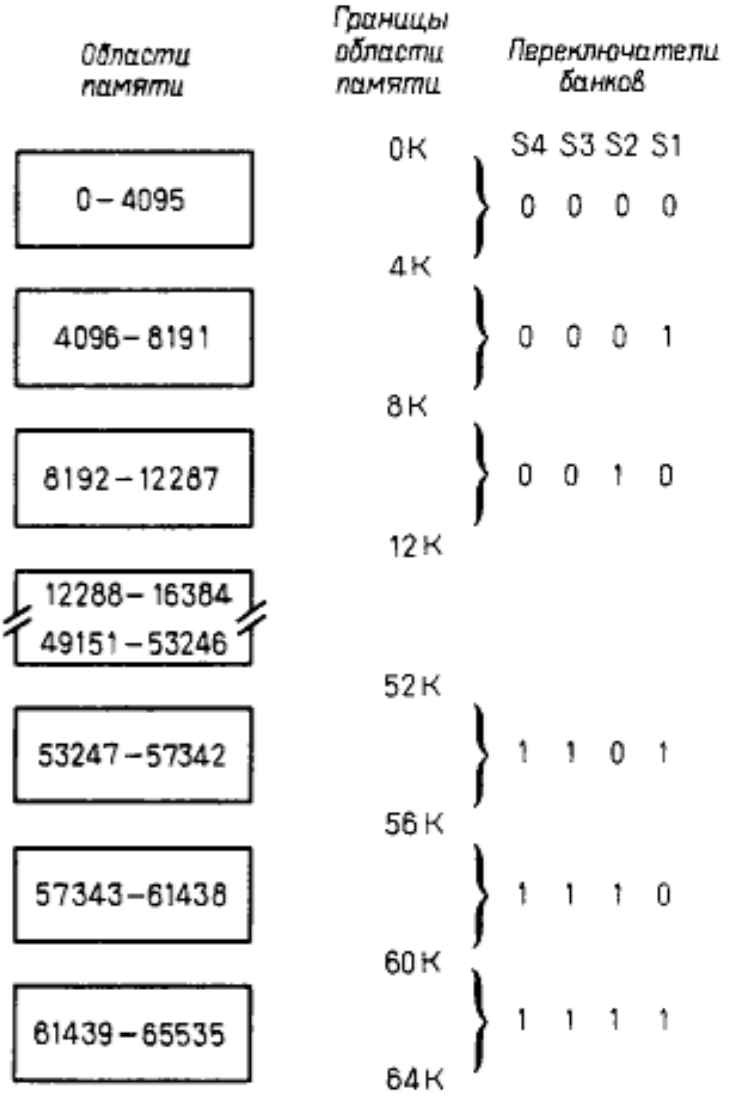


Рис. 11.16. Состояние переключателей банков (к ответу на вопрос п. 19 заданий для самопроверки).

Ответы на вопросы заданий для самопроверки

1. г. 2. в. 3. б. 4. а. 5. б. 6. в. 7. а. 8. б. 9. в. 10. г. 11. в. 12. г. 13. а.

14. Одним из стробирующих сигналов загружается дешифратор строки, другим – дешифратор столбца. Это усложнение обработки адреса позволяет сэкономить число внешних выводов микросхемы, предназначенных для передачи адреса.

15. В статических ОЗУ данные запоминаются в триггерных схемах. В динамических ОЗУ данные хранятся в виде зарядов входных емкостей транзисторов.

16. Если не регенерировать периодически содержимое динамического ОЗУ, то входные емкости транзисторов разряжаются, что приводит к потере данных.

17. Схемы выбора кристалла и дешифратора адреса.

18. Относительная доля накладных расходов снижается, потому что при удвоении объема памяти в данном случае накладные расходы совсем не возрастают или возрастают весьма незначительно.

19. См. рис. 11.16.

20. Потому что наличие логической схемы регенерации увеличивает накладные расходы при построении памяти.

21. Этот сигнал может быть, например, использован для запуска логической схемы, вырабатывающей стробирующие сигналы строки и столбца.

22. г. 23. б. 24. а. 25. в.

26. Потому что его содержимое может быть легко изменено при обнаружении ошибок.

27. Микро-ЭВМ не может выполнять никаких

действий, не получая команд. В ПЗУ должны находиться команды, которых достаточно, чтобы микропроцессор мог осуществить ввод данных из внешнего запоминающего устройства.

28. Потому что микропроцессор должен располагать какой-то оперативной памятью для записи в нее данных в процессе работы.

29. Скорость ПДП – 1 000 000 пересылка/с, скорость считывания информации с ленты – 316,2 байт/см × 189,75 см/с = 60 000 пересылка/с. Скорость пересылки ограничивается параметрами устройства хранения информации на магнитной ленте.

30. Скорость считывания информации с ленты равна 632,4 байт/см × 316,25 см/с = 200 000 пересылка/с. Скорость пересылки ограничивается параметрами устройства хранения информации на магнитной ленте.

31. Потому что в ходе каждой пересылки выполняется по сути дела чтение информации ввода-вывода, а также изменение количества оставшихся циклов. Кроме того, скорость пересылки ограничивается длительностью циклов выполнения команд микропроцессора, а не только значением цикла памяти.

32. Четыре регистра служат для размещения начальных адресов файлов и четыре – для подсчета числа переданных байтов.

Глава 12.
Система ввода-вывода

В настоящей главе рассматриваются способы ввода данных в микропроцессор и способы их вывода оттуда. В предыдущих главах мы получили представление о команде ввода IN, которая использовалась для передачи данных из входного порта в аккумулятор микропроцессора. В настоящей главе мы ознакомимся с реализацией команды вывода OUT. Кроме того, рассмотрим ряд распространенных устройств ввода-вывода, используемых для связи с микропроцессорными системами, и два устройства памяти большой емкости.

Здесь же приведен пример порта параллельного ввода-вывода и указаны способы управления. (Управление портом параллельного ввода-вывода организовано подобно тому, как реализовано управление памятью.) Рассмотрены средства организации последовательной передачи данных, которые применяются для связи микропроцессора с удаленными устройствами; при этом обычно используется код ASCII. Описаны также управляющие сигналы, необходимые для управления техническими средствами, обеспечивающими последовательную связь. Дано представление о преобразователях параллельного кода в последовательный и последовательного кода в параллельный.

Примеры использования команд ввода-вывода при работе с внешними устройствами были рассмотрены выше. В данной главе мы ознакомимся с другими способами, с помощью которых в микропроцессорных системах можно адресовать устройства ввода-вывода. Если внешнее устройство подключено к микропроцессору, то оно рано или поздно потребует «внимания» микропроцессора. Изучим два способа, с использованием которых внешнее устройство может «привлечь внимание» микропроцессора. Один из способов предполагает использование программы опроса, второй реализуется микропроцессорной системой прерываний.

12.1. Устройства ввода-вывода

Для ввода данных в микропроцессор и для их вывода используются устройства различного типа. Выбор устройства ввода-вывода зависит от источника информации, с которым работает микропроцессор. Например, если данные предполагается хранить на магнитном носителе, то в качестве устройства ввода-вывода может быть выбран либо накопитель на магнитной ленте, либо накопитель на магнитном диске. Для связи человека с ЭВМ, по-видимому, наиболее подходящими устройствами ввода и вывода данных являются клавиатура и дисплей. В этом разделе мы рассмотрим, каким образом осуществляется связь различных устройств с микропроцессором.

Прежде всего разделим устройства ввода-вывода на две группы. К первой группе отнесем устройства, используемые для связи человека с ЭВМ. Во вторую группу включим устройства, связывающие микропроцессор с техническими средствами. Легко представить себе, что устройства, используемые для связи человека с машиной, дол-

жны обладать особыми характеристиками. Эти устройства должны обеспечивать передачу обычных арифметических знаков операций, нескольких специальных математических символов и знаков пунктуации. Обычно процесс взаимодействия человека с машиной протекает медленно. При этом, как правило, для ввода данных используется клавишное устройство, а для отображения информации – дисплей.

Связь микропроцессора с другими устройствами, например с накопителями на магнитных дисках, производится с использованием различных кодов. Ввод и вывод данных осуществляются с высокой скоростью. В большинстве случаев данные вводятся в микропроцессор в той же форме, что и выводятся оттуда.

Наиболее распространенным устройством, с помощью которого человек вводит данные в микропроцессор, является клавишное устройство ввода. Известны клавишные устройства трех типов. К первому типу этих устройств относится простая клавиатура для ввода цифр. Такую клавиатуру имеют обычные электронные калькуляторы.

Клавиатура второго типа предназначена для ввода полного набора буквенно-цифровых символов. Обычно такой клавиатурой снабжаются оконечные устройства ЭВМ. Она позволяет вводить все буквенные и цифровые символы (буквы как прописные, так и строчные) и 20–30 специальных символов. К специальным символам относятся знаки математических операций, знаки пунктуации и небольшое количество управляющих символов.

Третий тип клавиатуры – это специальная клавиатура. Необходимость применения такой клавиатуры часто возникает в практике проектирования различных систем. Например, в системе управления кондиционированием воздуха нецелесообразно применять буквенно-цифровую клавиатуру. В этом случае предпочтительнее иметь клавиши с надписями: КОНДИЦИОНЕР ВКЛЮЧЕН, ОБОГРЕВ, ВЕНТИЛЯТОРЫ, НАСОСЫ и т. п.

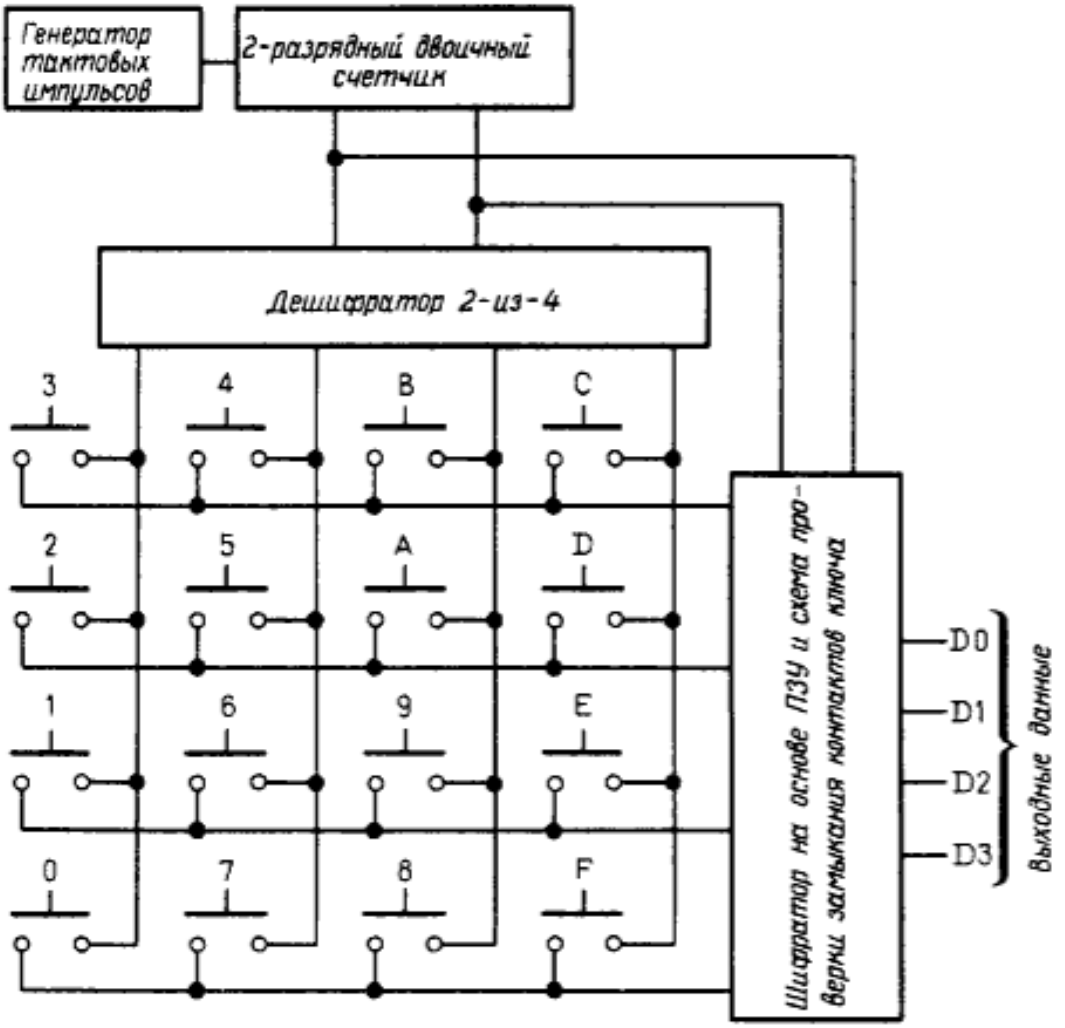
Большинство клавиатур построено по одному и тому же принципу. Упрощенное схематическое изображение клавиатуры представлено на рис. 12.1. На этом рисунке изображена обычная матрица, имеющая вертикальные линии (колонки) и горизонтальные линии (строки). Колонки клавиатуры сканируются, т. е. на них последовательно подаются сигналы. Сначала сигнал подается на первую колонку, затем на вторую, третью и, наконец, на четвертую колонку. Затем эти действия повторяются.

Сканирование может выполняться самим микропроцессором, оно может осуществляться и с помощью отдельной интегральной схемы – сдвигового регистра. Для этой цели можно использовать также двоичный счетчик и дешифратор (рис. 12.1). Вертикальные линии матрицы клавиатуры связаны с горизонтальными линиями матрицы с помощью ключей. В рассмотренном примере используется 16 ключей. Такое количество ключей соответствует матрице 4×4.

Линии, по которым поступают сигналы сканирования, и горизонтальные линии матрицы клавиатуры связаны со схемой кодирования. Эта схема выполняет две функции. Сначала она несколько раз подряд проводит проверку замыкания контактов ключа. Это делается для того, чтобы быть уверенным, что замыкание контактов действительно имеет место. Многократная проверка позволяет избежать появления ошибок. Такой прием называют защитой от ложных срабатываний клавиатуры, обусловленных эффектом дребезга контактов. Так, если 10 раз подряд было установлено, что один и тот же контакт находится в замкнутом состоянии, то велика вероятность того, что определенная вертикальная линия и определенная горизонтальная линия действительно соединены друг с другом посредством ключа. Таким образом делается вывод, что клавиша клавиатуры, соответствующая пересечению вертикальной и горизонтальной линий матрицы клавиатуры, нажата.

После того как произведена проверка надежности замыкания контактов, может быть выполнено кодирование выходных данных. Теперь остановимся на схеме кодирования. Эта схема представляет собой ПЗУ. Она воспринимает информацию, поступающую на вертикальные и горизонтальные линии матрицы клавиатуры, и образует необходимые выходные сигналы, которые могут быть представлены в параллельной или последовательной форме.

Аппаратные средства, обеспечивающие



Ключ	Выходные данные			
	D0	D1	D2	D3
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

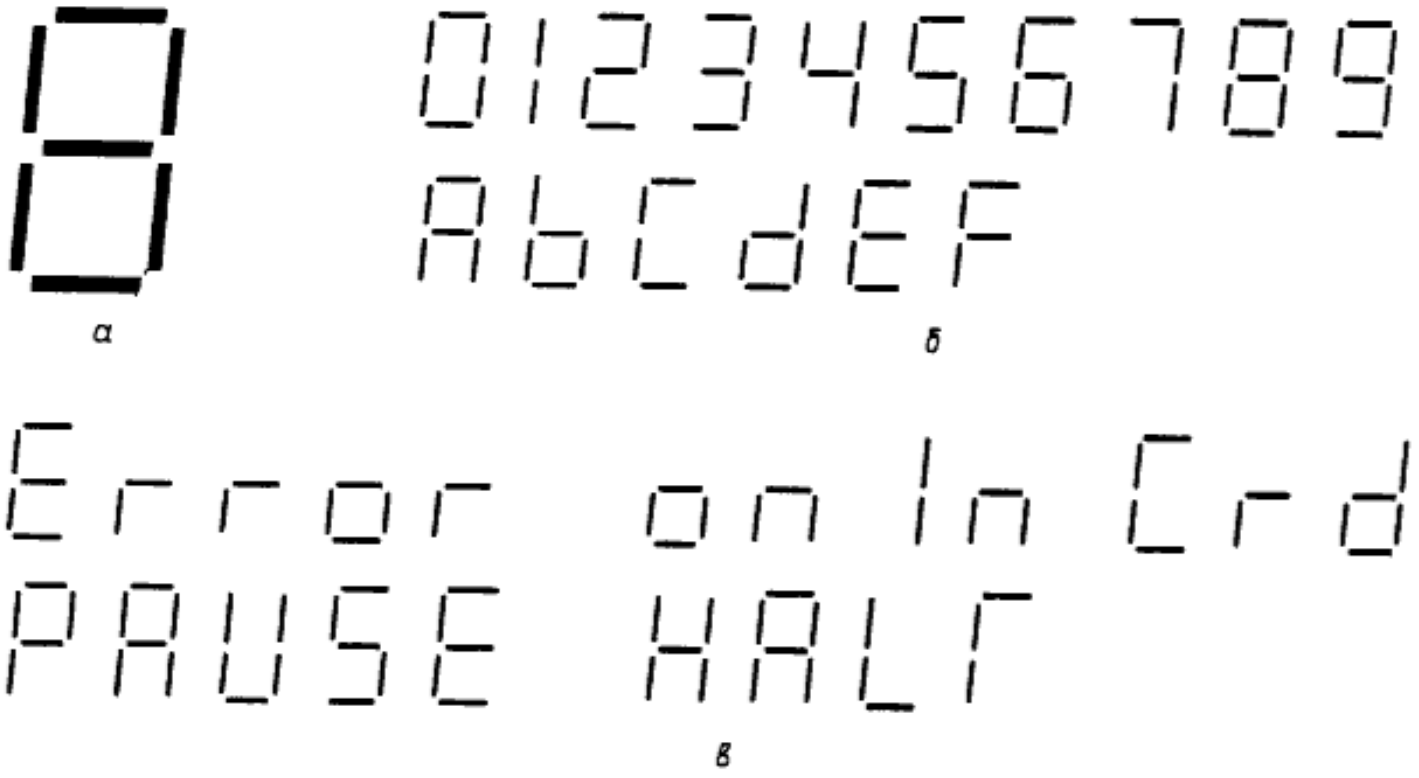
Рис. 12.1. Схема клавиатуры для ввода шестнадцатеричных цифр.

На вход двоичного счетчика подаются синхронизирующие импульсы частотой 400 Гц, поэтому полное сканирование клавиатуры производится за 10 мс. Прежде чем данные поступят на выход, с помощью специальной логической схемы производится трехкратная проверка наличия замыкания контактов одного и того же ключа

функционирование клавиатуры микропроцессорных систем, обычно реализуются одним из двух способов. При первом подходе к реализации аппаратных средств клавиатуры как сканирование вертикальных линий клавиатуры, так и формирование выходных сигналов клавиатуры выполняется самим микропроцессором. Процесс сканирования вертикальных линий матрицы очень простой клавиатуры может осуществляться микропроцессором, на основе которого построена микро-ЭВМ. Если используется такое решение, то матрица клавиатуры должна соединяться с портами ввода-вывода микропроцессора. Для выполнения сканирования вертикальных линий, выполнения процедуры многократного подтверждения факта замыкания контактов ключа и для формирования выходных сигналов клавиатуры в этом случае требуется специальная подпрограмма. Для обеспечения указанных функций для очень сложных клавиатур может быть использован и специально выделенный для данной цели микропроцессор. Второй способ реализации клавиатуры заключается в использовании для выполнения этих функций специальных интегральных схем. Так как клавиатуры широко используются для взаимодействия пользователя с техническими средствами, был разработан ряд больших интегральных схем.

В состав систем, имеющих клавиатуру, как правило, входит и устройство отображения информации (индикатор, дисплей и т. п.). Для отображения информации с помощью дисплеев используются буквенно-цифровые и специальные символы. Имеются три различных типа устройств отображения, которые аналогичны трем рассмотренным типам клавиатуры.

К первому типу устройств отображения информации относятся простые индикаторы, предназначенные для отображения только цифр. В таком индикаторе для отображения цифры может использоваться сегментная структура, состоящая, например, из



а *б* *в*

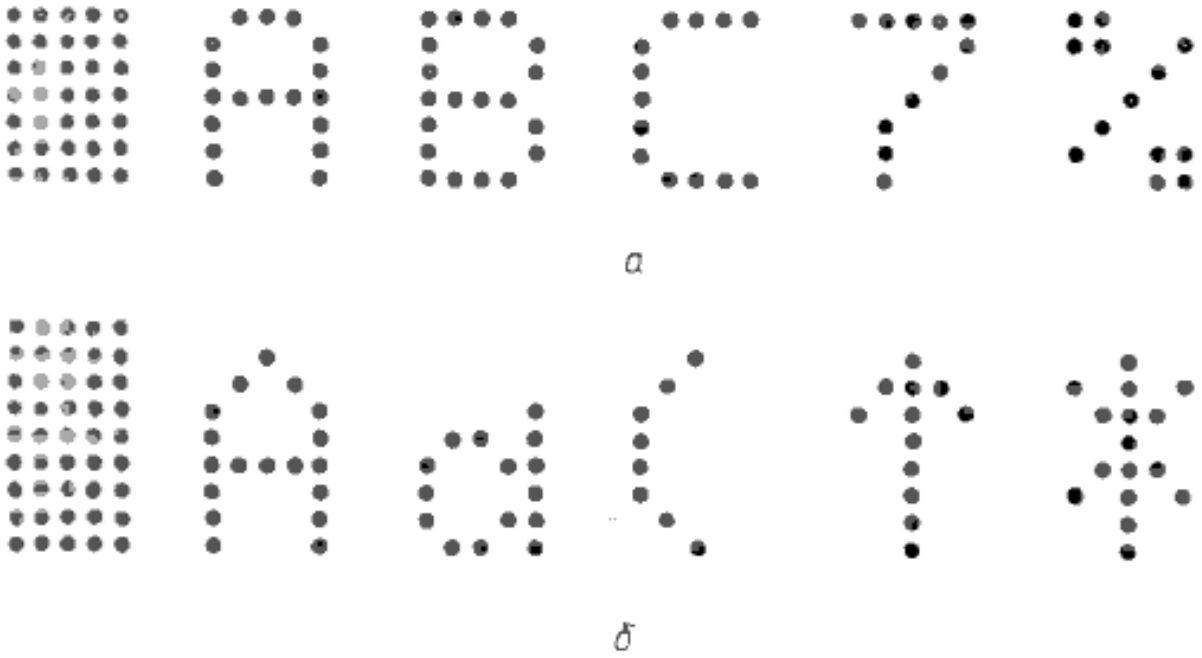
Рис. 12.2. Семисегментный дисплей, используемый для отображения информации. *а* – Базовая семисегментная структура; *б* – шестнадцатеричные цифры 0, 1, ..., F; *в* – примеры слов, написанных с помощью этого дисплея.

семи *сегментов*, или *элементов* (рис. 12.2, *а*). Индикаторы часто используются в калькуляторах и других устройствах, где необходимо отображать цифровую информацию. С помощью таких индикаторов можно отображать и ограниченный набор букв. Буквенные символы, применяемые для отображения шестнадцатеричных чисел, представлены на рис. 12.2, *б*. Используя различные комбинации из семи сегментов, с помощью цифрового индикатора можно отображать небольшое количество слов. Соответствующие примеры слов показаны на рис. 12.2, *в*.

В устройствах отображения второго типа используется точечная матрица 5 × 7 или точечная матрица 5 × 9. С помощью первой матрицы можно формировать большие буквенно-цифровые символы. Матрица второго типа позволяет формировать как прописные, так и строчные буквенные символы. На рис. 12.3 показаны примеры представления символов с помощью двух точечных матриц. На рис. 12.3, *а* изображены точечная матрица 5 × 7, прописные буквы A, B, C, цифра 7 и знак %, которые сформированы на этой матрице. Аналогично на рис. 12.3, *б* представлена точечная матрица 5 × 9 и показано, как будут выглядеть сформированные на ее основе символы A, a, (, ↑ и *.

Устройства отображения с точечными матрицами 5 × 7 и 5 × 9 могут быть построены в виде дисплеев на светоизлучающих диодах, на неоновых элементах и т. п. Весьма широко используются точечные матрицы в дисплеях на ЭЛТ. Хорошим примером такого устройства отображения является обычный дисплей, подключаемый к ЭВМ. Взглянув на экран такого дисплея, вы убедитесь в том, что символы, представляемые с помощью точечной матрицы, легко читаются.

К третьему типу относятся дисплеи специального назначения. В таких устройствах обычно отображаются не отдельные буквенно-цифровые символы, а специальные обозначения или сообщения, соответствующие функциям конкретной системы. Обратимся вновь к примеру системы управления кондиционированием, упомянутому выше. В такой системе вполне могли бы использоваться табло со следующими надписями: КОНДИЦИОНЕР ВКЛЮЧЕН, ПЕРЕГРЕВ ДВИГАТЕЛЯ, ВЫСОКИЙ УРО-



а

б

Рис. 12.3. Представление буквенно-цифровых символов с помощью *а*) точечной матрицы 5 × 7 и *б*) точечной матрицы 5 × 9.

ВЕНЬ ВОДЫ. НИЗКИЙ УРОВЕНЬ ВОДЫ или НАСОС ВКЛЮЧЕН.

Буквенно-цифровые устройства отображения можно также подразделить на устройства, обеспечивающие получение «твердой» и «мягкой» копии. Видеотерминалы, например, относятся ко второй группе и имеют клавиатуру и дисплей на ЭЛТ. Последовательно производятся как ввод данных в видеотерминал, так и вывод их оттуда. Экран дисплея может использоваться для отображения от 24 до 48 строк, в каждой из которых обычно содержится 40, 80 или 132 символа. Видеотерминалы часто называют ЭЛТ (так говорят, когда речь идет об описании самого дисплея, а не терминала) или терминалом.

По мере все более широкого распространения ЭВМ наряду с обычными терминалами стали появляться так называемые *интеллектуальные терминалы*. «Интеллектуальность» таких терминалов обычно связана с применением в них микропроцессоров. Интеллектуальный терминал может производить некоторые дополнительные операции по обработке данных и задание формата последних. Обычный терминал осуществляет передачу каждого символа, введенного с помощью клавиатуры, и отображение полученных символов. Терминал, реализующий такие функции, не является интеллектуальным. Однако его логические возможности можно развить посредством использования микропроцессора. Обычному терминалу не присущи, например, такие свойства интеллектуального терминала, как возможность изменения скорости передачи данных, проверка на четность или изменение формата путем применения специальных управляющих символов.

Другим распространенным типом буквенно-цифрового устройства вывода данных является печатающее устройство. Печатающее устройство дает «твердую копию» (распечатку на бумаге). В большинстве печатающих устройств до сих пор используется копировальная лента. В дешевых печатающих устройствах применяется точечная матрица 5 × 7 или 5 × 9; в дорогих печатающих устройствах используется печатающая головка с заранее подготовленным набором литер, подобная головкам пишущих машинок.

В микропроцессорных системах часто обеспечивается связь микропроцессора с внешними запоминающими устройствами большой емкости. Наиболее широкое распространение получили два вида магнитных запоминающих устройств большой емкости – магнитные ленты и гибкие магнитные диски. Указанные устройства имеют большой объем памяти, информация в которой сохраняется при выключении электропитания. Накопители на магнитных лентах имеют сравнительно низкую стоимость, а устройства памяти на гибких дисках оеспе-

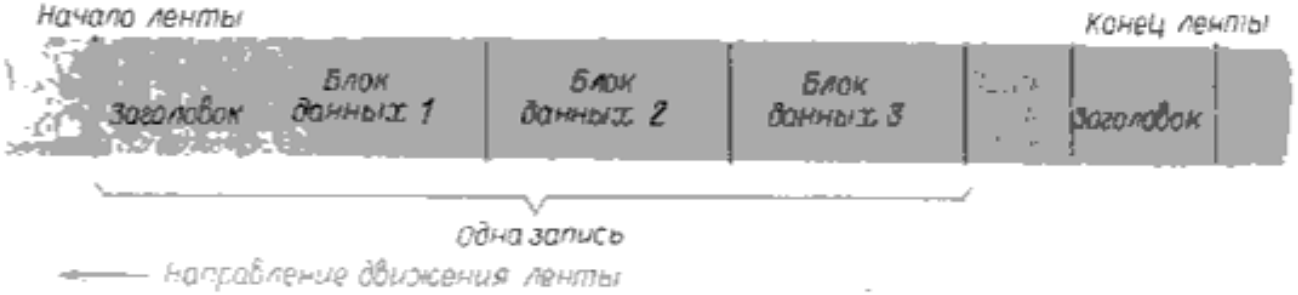


Рис. 12.4. Формат записи данных на магнитной ленте. (Заголовок содержит такую информацию, как имя записи, длина записи и т. п.)

чивают относительно малое время доступа к данным.

В микропроцессорных системах для записи данных широко используются стандартные кассеты фирмы Philips или кассеты несколько большей емкости. Обычно на магнитной ленте, содержащейся в такой кассете, может храниться от 250 000 до 1 000 000 байт данных. Взаимодействие микропроцессора с накопителем на магнитной ленте может осуществляться с использованием либо последовательного, либо параллельного интерфейса. Данные обычно записываются на магнитную ленту блоками. Так, если блок имеет размер 256 байт, то перед фактической записью информации на ленту потребуется заполнить в памяти соответствующий буфер полностью, т. е. поместить в него 256 байт данных. С целью последующей идентификации записи на магнитную ленту записывается специальная метка, или «заголовок».

Техника записи цифровых данных существенно отличается от техники записи звука, хотя внешне устройство цифровой записи на магнитную ленту и обычный бытовой магнитофон весьма схожи. Одно из главных отличий состоит в том, что устройство цифровой записи должно обладать возможностью предельно быстрого запуска и остановки процесса чтения-записи данных.

На рис. 12.4 представлен типичный формат записи на магнитной ленте. Магнитная лента является дешевым видом магнитной памяти большого объема, но допускает только последовательный доступ к данным. Поиск определенного блока данных может потребовать проведения поиска на всей длине ленты.

Гибкие магнитные диски лишены указанного выше недостатка, присущего магнитным лентам. Гибкий диск в бумажной оболочке (конверте) представляет собой гибкую круглую пластмассовую пластину, покрытую магнитным слоем (рис. 12.5, *а*); такой диск обычно имеет диаметр 127 или 203 мм. Щелевое отверстие в оболочке позволяет головке чтения-записи контактировать с магнитным материалом. На рис. 12.5, *б* показано расположение дорожек на гибком диске. Дорожки представляют собой концентрические кольца. Диск диаметром 203 мм имеет 77 дорожек для записи данных. Каждая дорожка, или кольцо, делится на 32 сектора.

Скорость вращения гибкого диска в рабочем режиме составляет 360 об/мин. Таким образом, время полного оборота равно 166 мс. Головка чтения-записи может быть установлена над любой из 77 дорожек. После того как головка установлена над заданной дорожкой, время доступа к данным, находящимся на этой дорожке, не превышает 166 мс.

Среднее время доступа к данным в пределах одной дорожки гибкого диска составляет $166/2 = 83$ мс. Полное время доступа к данным на гибком магнитном диске зависит от того, насколько быстро головка подводится к определенной дорожке, и от скорости вращения диска. Типичное среднее время доступа к данным, расположенным на магнитном диске, равно $\sim 1{,}25$ с. Следует отметить, что для распространенных типов накопителей на магнитной ленте время доступа к данным составляет 20 или 30 с.

Емкость каждого сектора на гибком магнитном диске равна 128 байт. Следовательно, 32-секторная дорожка имеет емкость 4096 байт. В микропроцессорных системах с гибкими дисками обмен между накопителями на магнитных дисках и основной памятью часто производится блоками, размер которых выбирается равным размеру дорожки.

В системах памяти на гибких магнитных

FFFA	FFFB	Шестнадцатеричный код	Прерывание
0000 0100	0000 1000	0408	I0
0000 0100	0001 0000	0410	I1
0000 0100	0001 1000	0418	I2
0000 0100	0010 0000	0420	I3
0000 0100	0010 1000	0428	I4
0000 0100	0011 0000	0430	I5
0000 0100	0011 1000	0438	I6
0000 0100	0100 0000	0440	I7

Рис. 12.26. Таблица векторов, генерируемых системой, представленной на рис. 12.25.

Рассматриваемый здесь гипотетический микропроцессор имеет три входа для подачи запросов на прерывание. Эти входы показаны на схеме, представленной на рис. 12.27.

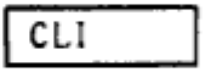
Запросу на прерывание, подаваемому на вход «Начальная установка», соответствует вектор, расположенный в области памяти с адресами FFFE и FFFF. Прерывание, обусловленное подачей сигнала на вход «Начальная установка», имеет наивысший приоритет. Сигнал начальной установки, или сброса, микропроцессора подается в тех случаях, когда в силу каких-либо причин, например после сбоя, требуется возобновить работу микропроцессора с некоторого начального состояния. Большинство микропроцессоров после включения питания начинают свою работу с выполнения команды «Сброс». Часто при выполнении этой команды управление передается некоторой вспомогательной программе, которая производит пересылку главной программы из внешней памяти в ОЗУ микропроцессора.

Следующий уровень приоритета в микропроцессоре имеет немаскируемое прерывание. Такому прерыванию соответствует вектор, содержащийся в областях памяти с адресами FFFC и FFFD. На третий вход подаются запросы на прерывание, которые могут быть маскированы. Маскируемому прерыванию соответствует вектор, содержащийся в областях памяти с адресами FFFA и FFFB. Единственное отличие маскируемого прерывания от немаскируемого состоит в том, что в первом случае программист может разрешить или запретить микропроцессору реагировать на маскируемое прерывание. Чтобы маскировать прерывание, программист должен предусмотреть запись значения 1 в разряд «Прерывание» регистра состояния микропроцессора.

Для установки и сброса маски прерывания предусмотрены следующие специальные команды:

СБРОС МАСКИ ПРЕРЫВАНИЯ

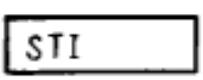
CLI 0 → Разряд «Прерывание»



1 байт, два цикла

УСТАНОВИТЬ МАСКУ ПРЕРЫВАНИЯ В 1

STI 1 → Разряд «Прерывание»



1байт, два цикла

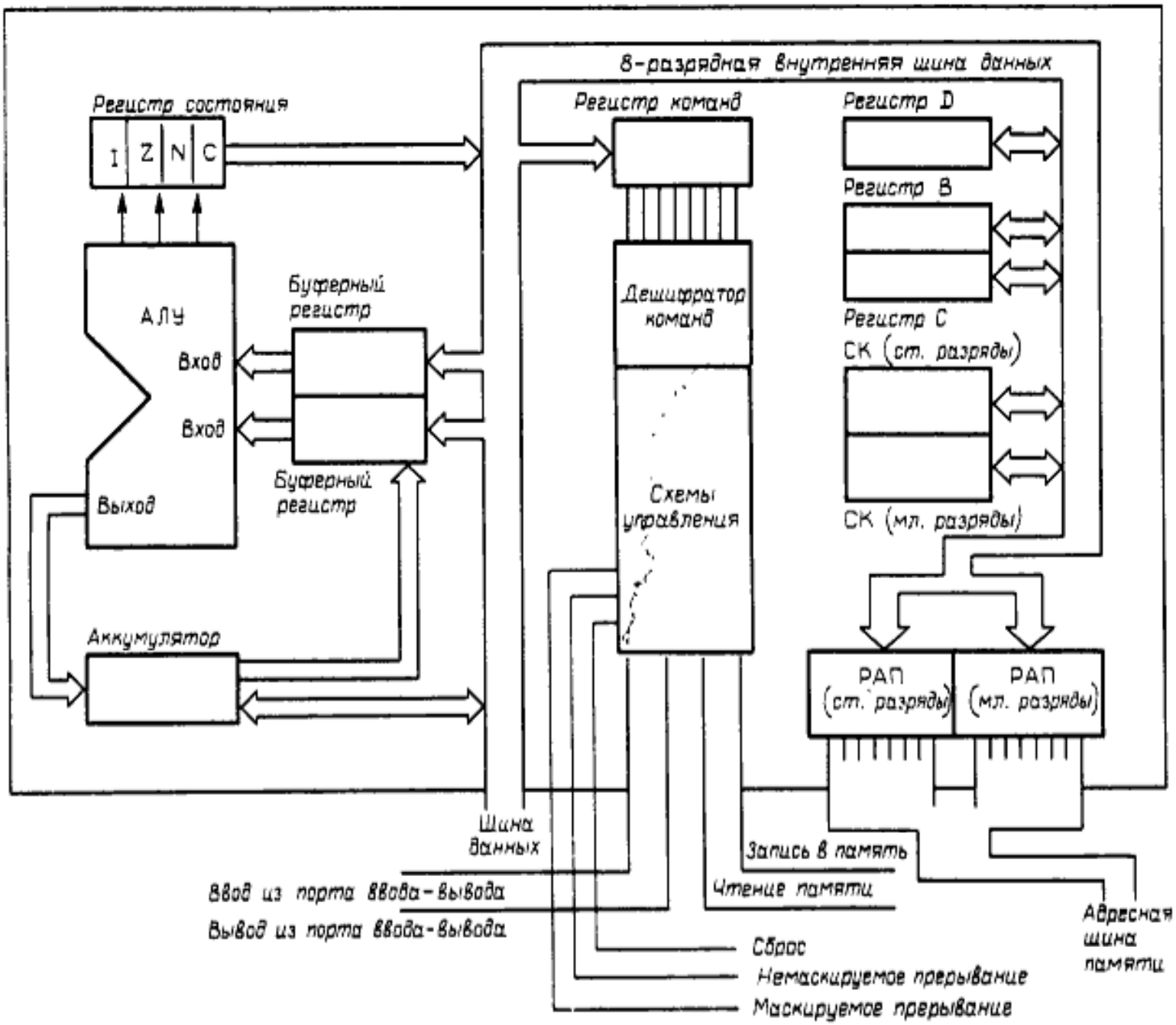


Рис. 12.27. Схема микропроцессора, имеющего три входа для подачи запросов на прерывание.

Если маска прерывания не установлена, т. е. разряд «Прерывание» в регистре состояния микропроцессора имеет значение 0, то при поступлении запросов на прерывание микропроцессор будет реагировать на эти запросы и выполнять обработку прерываний. Если же маска прерывания установлена в 1, т. е. разряд «Прерывание» имеет значение 1, микропроцессор не будет реагировать на сигналы, поступающие на вход маскируемого прерывания.

Часто маску прерывания устанавливает в самом начале своего выполнения программа обработки прерывания. Тем самым предотвращается возможность других прерываний до тех пор, пока не завершится выполнение программы обработки текущего прерывания. Маска прерывания снимается на том этапе выполнения программы обработки прерывания, когда уже могут быть разрешены другие прерывания.

Значение разряда «Прерывание» регистра состояния не оказывает никакого влияния на прием запросов на немаскируемые прерывания и их обработку. При поступлении запроса на немаскируемое прерывание микропроцессор выбирает вектор прерывания непосредственно из областей с адресами FFFC и FFFD.

Задания для самопроверки

30. Если для обнаружения устройства, требующего обслуживания, используется способ опроса, то какое из указанных ниже действий должно выполнять устройство ввода-вывода: а) осуществить прерывание работы микропроцессора; б) устанавливать приоритет прерывания; в) ждать запуска программы опроса; г) работать со скоростью, не превышающей 110 бод?

31. Каким из перечисленных ниже факторов определяется приоритет программы опроса:

а) приоритетом прерывания; б) местом подключения устройства к шине микропроцессора; в) скоростью работы устройства ввода-вывода; г) программными средствами?

32. Подобно чему [а) аппаратному вызову подпрограммы, б) команде безусловного перехода, в) команде условного перехода, г) команде «Сброс»] выполняется прерывание?

33. Укажите, какие из прерываний [а) маскируемое прерывание, б) немаскируемое прерывание, в) начальная установка, г) вектор прерывания] не воспринимаются микропроцессором, если разряд «Прерывание» регистра состояния микропроцессора имеет значение логической 1.

34. Перечислите три основных действия, выполняемые микропроцессором после поступления на вход запроса на прерывание.

35. Почему при обработке прерывания, обусловленного подачей сигнала на вход «Начальная установка», содержимое счетчика команд не загружается в стек?

Упражнения

12.1. Если в клавишном устройстве применен полный код ASCII, то с его помощью можно вводить 10, 64, 128 или 256 символов?

12.2. Сравните магнитные ленты и гибкие магнитные диски по следующим параметрам:

а) емкости памяти;

б) времени доступа;

в) стоимости;

г) возможности последовательного доступа к данным.

12.3. Какие указанные ниже средства применяются для формирования символа в дисплее на ЭЛТ, позволяющем отображать как прописные, так и строчные буквы:

а) точечная матрица 5 × 7;

б) точечная матрица 5 × 9;

в) матрица 4 × 4;

г) дисплей на 128 × 24 символа?

12.4. Укажите, какие элементы [бумага; точечные матрицы, предназначенные для печати символов в виде совокупности точек; печатающая головка, подобная головке обычных пишущих машинок] используются в печатающих устройствах.

12.5. Для отображения каких символов [цифр и некоторых букв, только цифр (0–9), только шестнадцатеричных цифр (0–F) или всех прописных букв] может использоваться семисегментная структура?

12.6. Какие перечисленные ниже элементы входят в состав порта параллельного ввода-вывода:

а) параллельные выходы;

б) параллельные входы;

в) регистр состояния?

12.7. Для каких указанных ниже систем организация ввода-вывода по аналогии с обращением к памяти не является оправданной:

а) система с ограниченным объемом памяти;

б) система с большим количеством устройств ввода-вывода;

в) система, выполняющая суммирование данных, поступающих из двух портов ввода-вывода;

г) система, в которой реализован режим прямого доступа к памяти?

12.8. Является ли шина данных IEEE-488 внутренней адресной шиной, внутренней шиной данных, внешней параллельной шиной или программируемым устройством?

12.9. Для связи каких указанных ниже элементов предназначен асинхронный приемопередатчик:

а) модем с портом параллельного ввода-вывода;

б) линия последовательной передачи данных с портом параллельного ввода-вывода;

в) порт параллельного ввода-вывода с линией последовательной передачи данных?

12.10. Основным логическим элементом в универсальном приемопередатчике является сумматор, логическая схема ИСКЛЮЧАЮЩЕЕ ИЛИ, сдвиговый регистр или регистр состояния?

12.11. Быстродействие приемопередатчика, выраженное в бодах, равно а) скорости передачи данных по линии, умноженной на 16; б) скорости передачи сигналов или в) длине слова данных?

12.12. Укажите двоичный код ASCII для каждого из перечисленных ниже символов. Составьте таблицу, содержащую символы, двоичные семиразрядные коды ASCII и соответствующие значения восьмого разряда, используемого как разряд контроля по нечетности:

а) управляющий символ «Конец текста», б) E,

в) F, г) W, д) подача бумаги на одну строку, е) возврат каретки, ж) пробел, з) M, и) T, к) 8, л) 15, м) Ø.

12.13. Объясните, в чем состоит ошибка кадрирования (синхронизации).

12.14. Каким образом распознает принимающий приемопередатчик стартовый бит?

12.15. Предположим, что передатчик универсального приемопередатчика работает со скоростью 150 бод. Однако приемник рассчитан на прием данных, поступающих со скоростью 300 бод. Если при этом передается символ 5, представленный в коде ASCII, и соответствующий разряд контроля по четности, то какую информацию примет приемопередатчик?

12.16. Какие из следующих ниже условий не имеют никакого отношения к регистру состояния приемопередатчика:

а) наложение;

б) контроль;

в) число стоп-битов;

г) данные готовы?

12.17. Каково назначение операции пересылки содержимого регистра состояния УАПП в выходной буфер последнего:

а) записать содержимое регистра в память;

б) выполнить проверку содержимого регистра в аккумуляторе;

в) установить в 0 разряд «Наложение» регистра состояния;

г) выполнить проверку значения контрольного разряда?

12.18. Укажите, какие из перечисленных ниже средств предусмотрены в приемопередатчике, подключаемом непосредственно к микропроцессору, и вместе с тем не используются в автономных приемопередатчиках:

а) регистр состояния;

б) вход синхронизирующих сигналов;

в) дешифратор адреса;

г) последовательный выход?

12.19. Объясните, насколько точно название «Универсальный асинхронный приемопередатчик» характеризует соответствующее устройство.

12.20. Опишите простую программу, которая могла бы синхронизировать работу УАПП и микропроцессора.

12.21. Чем объясняется более широкое распространение стандарта RS-232 на средства последовательной передачи данных по сравнению с передачей сигналов с помощью стартстопного телеграфного аппарата, имеющего источник тока 20 мА?

12.22. С какой целью в линиях последовательной передачи данных используются оптоизоляторы (оптроны)?

12.23. Опишите принцип действия модема и условия его применения.

12.24. Изобразите блок-схему программы опроса, в которой уровни приоритета портов ввода-вывода определяются следующей последовательностью номеров портов: 1, 3, 2.

12.25. Если приемопередатчик может подавать сигналы запроса на обслуживание на вход маскируемых прерываний микропроцессора, то каким образом можно установить запрет на эти прерывания?

12.26. Каково назначение схемы кодирования, подобной схеме, изображенной на рис. 12.25?

12.27. Изобразите в виде блок-схемы последовательность действий, выполняемых микропроцессором с момента поступления запроса на прерывание и до возврата в прерванную программу.

12.28. Объясните значение понятия «вектор прерывания».

12.29. Шестнадцатиразрядный микропроцессор получает от УАПП символы в коде ASCII с разрядом контроля по четности. Принимаемые данные записываются в основную память. Почему в программе записи содержится много команд сдвига?

12.30. В вашем распоряжении микро-ЭВМ, связанная с другими ЭВМ синхронно управляемым каналом передачи данных, обеспечивающим передачу данных со скоростью 1 М бод. Могли бы вы применить в этой микро-ЭВМ программируемую передачу данных или прямой доступ к памяти?

Ответы на вопросы заданий для самопроверки

1. г. 2. б. 3. г. 4. в.

5. Если для каждого ключа должен быть предусмотрен отдельный вход ПЗУ, то в случае клавиатуры с большим количеством клавиш потребуется слишком много входов ПЗУ.

6. Смогли бы. Примеры отображения заданных слов представлены на рис. 12.28.

7. а) По-видимому, смогли бы, так как мы ожидаем появления определенных слов и, следовательно, сможем правильно прочитать слова с нестандартным изображением некоторых букв; б) мы испытывали бы большие неудобства при чтении таких слов.

8. б. 9. в. 10. г. 11. а.

12. Недостатком является то, что при использовании этого метода расходуется память. Основное достоинство состоит в возможности использования различных команд микропроцессора непосредственно при вводе или выводе данных.

13. Этот способ можно использовать, не принимая во внимание имеющиеся линии ввода-вывода и команды ввода-вывода.

14. Программируемая передача данных допускается, однако при этом данные будут пересылаться в аккумулятор, а не в область памяти.

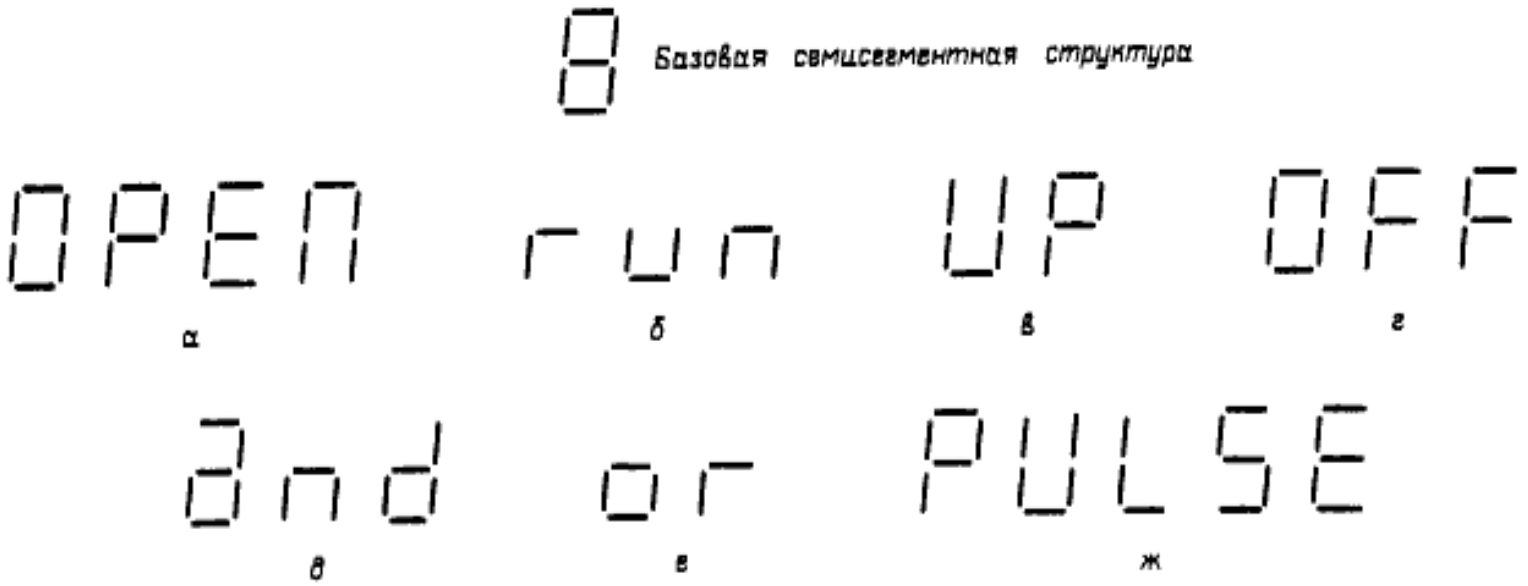


Рис. 12.28. Ответ на вопрос п. 6 заданий для самопроверки.

15. а) Принимает информацию; б) принимает и выдает; в) принимает и выдает; г) принимает; д) является контроллером; е) принимает информацию; ж) принимает; з) принимает и выдает; и) принимает и выдает; к) принимает и выдает. 16.г. 17.в. 18.а. 19.б. 20.в.

21. а) УАПП – универсальный асинхронный приемопередатчик – устройство, предназначенное для приема и передачи последовательно передаваемых данных. Он преобразует данные из последовательной формы в параллельную, из параллельной в последовательную, а также выполняет функции контроля и управления.

б) В бодах выражается скорость передачи сигналов для одного символа, передаваемого по линии последовательной передачи данных. Бод соответствует 1 бит/с.

в) ASCII (American Standard Code for Information Interchange) – это Стандартный американский код для обмена информацией. Он содержит буквенно-цифровые символы, знаки пунктуации и математических операций.

г) Сигнал синхронизации используется для управления работой приемопередатчика. Частота этого сигнала численно равна скорости передачи в бодах, умноженной на 16.

д) «Наложение» – условие, при котором некоторый символ, передаваемый по линии последовательной передачи данных, «набегает» на предыдущий символ.

е) Ошибка кадрирования состоит в том, что приемопередатчик ошибочно определяет стартовый бит.

ж) Стартовый бит является первым в группе битов, соответствующих одному передаваемому символу. При последовательной передаче сигналов появление стартового бита определяется изменением сигнала от уровня логического 0 до уровня логической 1.

з) Контрольным битом снабжается кодовая комбинация, соответствующая передаваемому символу. Значение контрольного бита должно быть таким, чтобы число единиц в передаваемой кодовой комбинации было четным, если используется контроль по четности, или нечетным, если используется контроль по нечетности. Наличие контрольного бита позволяет обнаруживать на приемном конце некоторые ошибки, возникающие при передаче сигналов.

и) Команда JNE (Jump If Not Equal) – ПЕРЕХОД, ЕСЛИ НЕ РАВНО – часто используется после команды сравнения. Согласно этой команде, проверяется значение разряда нулевого результата в регистре состояния микропроцессора. Если этот разряд имеет значение «логический 0», то осуществляется переход.

22.

	Шестнадцатеричный код символа	Код символа с контрольным битом
а.	36	36
б.	41	41
в.	5E	DE
г.	4C	CC
д.	70	F0
е.	34	B4
ж.	2A	AA
з.	61	E1
и.	30	30
к.	07	87
л.	24	24
м.	62	E2
н.	04	84
о.	20	A0

23. в. 24. а. 25. а. 26. б.
27. 1) д, 2) г, 3) б, 4) а, 5) в.
28. *а* – стартового, *б* – 1, *в* – 0, *г* – данных.
29. Дело в том, что при рассматриваемом способе передачи не известно, когда поступит следующий символ. Следовательно, интервал времени между моментом окончания передачи некоторого символа и моментом начала передачи следующего символа является случайной величиной.
30. в. 31. г. 32. а. 33. а.
34. а) Микропроцессор завершает выполнение текущей команды; б) текущее значение счетчика команд загружается в стек; в) в счетчик команд помещается вектор прерывания, и начинается выполнение программы обработки прерывания.
35. В этом нет необходимости, так как начальная установка микропроцессора выполняется при подаче питания, а также в любое время, когда потребуется, чтобы микропроцессор начал выполнять программу с команды, находящейся по определенному адресу. Перед выполнением начальной установки микропроцессора счетчик команд может иметь произвольное значение.

Глава 13. Дополнительные способы адресации

В настоящей главе рассматриваются еще три способа адресации, применяемые в микропроцессорах при выборке содержимого областей памяти. В большинстве команд реальных программ используется обращение к областям памяти. Фирмы-изготовители микропроцессоров уделяют большое внимание реализации различных способов обращения к памяти.

Рассматриваемые здесь способы адресации памяти следующие: с индексированием; относительный и способ адресации данных, хранимых в стеке, с помощью указателя стека. Во многих современных микропроцессорах используется либо первый, либо второй указанный способ. Почти каждый микропроцессор располагает стековым способом обращения к данным с помощью указателя стека.

Для обеспечения адресации с индексированием в микропроцессоре должен быть предусмотрен индексный регистр. Некоторые микропроцессоры имеют несколько индексных регистров. Будет рассмотрено, каким образом может быть преобразована программа, в которой используется адресация с индексированием, в программу, которая могла бы выполняться микропроцессором, не имеющим индексного регистра.

Обсуждаются также способы манипулирования с указателем стека в операциях со стеком. Нам уже известно, каким образом используются указатель стека и стек при выполнении обращения к подпрограммам и при возврате из них, а также при обработке прерываний. В настоящей главе, кроме того, рассматриваются приемы использования указателя стека для обеспечения доступа к желаемым областям памяти. Описываются команды, которыми может воспользоваться программист для работы со стеком.

13.1. Адресация с индексированием

На рис. 13.1 представлена видоизмененная структурная схема 8-разрядного микропроцессора. На этой схеме вместо ранее используемой регистровой пары BC используется 16-разрядный индексный регистр. Очевидно, что схема стала немного проще.

Для выполнения операций с индексным регистром в данном случае можно использовать следующие четыре команды: ЗАГРУЗКА НЕПОСРЕДСТВЕННАЯ ИНДЕКСНОГО РЕГИСТРА (команда, подобная команде загрузки непосредственных данных в регистровую пару), ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ ИНДЕКСНОГО РЕГИСТРА, ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ ИНДЕКСНОГО РЕГИСТРА и СРАВНЕНИЕ ИНДЕКСНОГО РЕГИСТРА С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ. Рассмотрим форматы указанных команд.

ЗАГРУЗКА НЕПОСРЕДСТВЕННАЯ ИНДЕКСНОГО РЕГИСТРА

LDX,Данные

Старший байт данных → $IX_{СБ}$
Младший байт данных → $IX_{МБ}$

LDX
МБ данных

3 байт, три цикла

Команды, предназначенные для выполнения операций над содержимым индексного регистра, аналогичны вышеописанным командам, используемым для выполнения операций над содержимым регистров. Команда загрузки применяется для записи в регистр желаемого значения. При выполнении команды загрузки индексного регистра разряд отрицательного результата в регистре состояния принимает значение 1, если в старшем 15-м разряде индексного регистра оказывается значение 1. Если в индексный регистр загружается нулевое значение, т.е. все 16 разрядов индексного регистра принимают значение логического 0, то разряд нулевого результата в регистре состояния равен 1.

Индексный регистр рассматривается как один 16-разрядный регистр, поэтому операции над отдельными байтами этого регистра не допускаются.

ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ ИНДЕКСНОГО РЕГИСТРА

INX $IX + 1 \rightarrow IX$

INX

1 байт, два цикла

ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ ИНДЕКСНОГО РЕГИСТРА

DEX $IX - 1 \rightarrow IX$

DEX

1 байт, два цикла

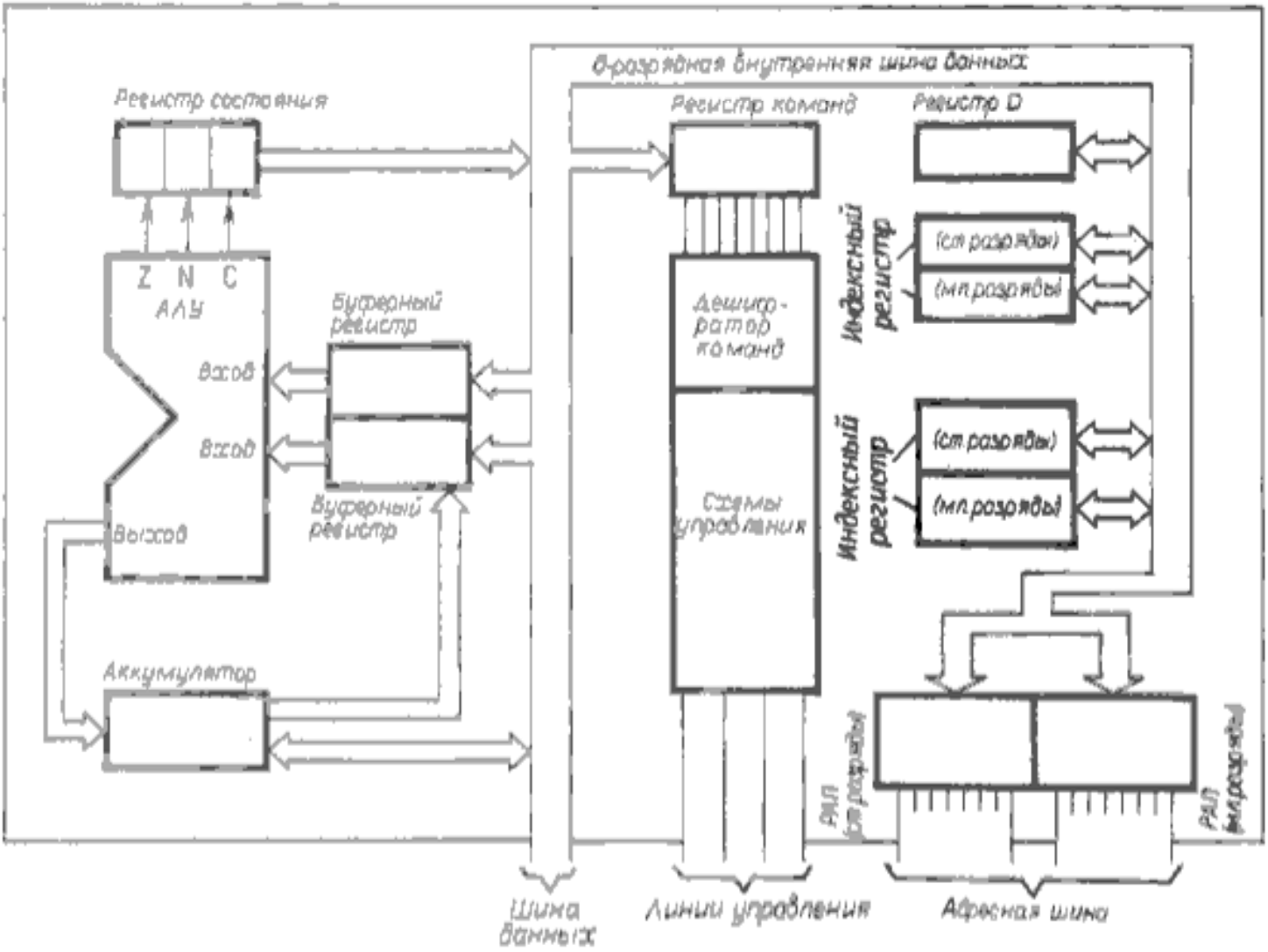
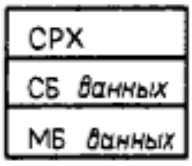


Рис. 13.1. Схема микропроцессора с индексным регистром.

Команды увеличения и уменьшения содержимого индексного регистра при использовании индексного регистра для модификации некоторого адреса позволяют обратиться к области с бóльшим или меньшим адресом памяти соответственно. При выполнении команд увеличения и уменьшения содержимого индексного регистра в регистре состояния может быть установлен в определенное состояние разряд нулевого результата. Этот разряд принимает значение, равное 1, если после увеличения или уменьшения содержимого индексного регистра во всех его 16 разрядах окажется значение, равное 0.

СРАВНЕНИЕ ИНДЕКСНОГО РЕГИСТРА С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ

CPX,Данные
$IX_{СБ}/IX_{МБ}$ – старший байт данных/младший байт данных



3 байт, четыре цикла

При выполнении команды сравнения используется 16-разрядное содержимое индексного регистра. Из содержимого индексного регистра вычитается 2-байтовый непосредственный операнд, содержащийся в команде сравнения. Как и во всех других командах сравнения, результат вычитания не записывается в память. В зависимости от результата вычитания в регистре состояния устанавливается соответствующее значение одного из следующих разрядов: нулевого результата, отрицательного результата или переноса. Отметим еще раз, что операция сравнения производится только над словами данных, т.е. она не может выполняться над отдельными байтами индексного регистра.

Индексный регистр применяется для реализации *адресации с индексированием*. Такой способ адресации осуществляется посредством сложения второго байта команды с содержимым индексного регистра. Полученная при этом сумма используется в качестве адреса области памяти.

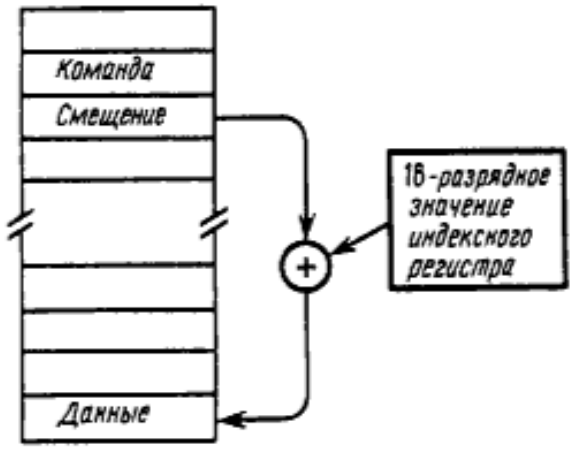


Рис. 13.2. Выполнение адресации с индексированием.

Рис. 13.2 дает представление о действиях, выполняемых при реализации способа адресации с индексированием. По существу это команда, при выполнении которой используется адресация по регистру и последующее вычисление адреса области памяти. Фактически путем сложения определяется новый 16-разрядный номер области.

Прибавляемая к содержимому индексного регистра 8-разрядная величина может принимать значения в диапазоне от 00 до FF. Эту величину называют смещением. Смещение может иметь любое значение в диапазоне $0_{10} - 255_{10}$. Следовательно, при некотором содержимом индексного регистра, выбирая значение смещения, можно адресовать 256 областей памяти.

Адресация с индексированием является новым для микропроцессоров способом адресации. Этот способ адресации может использоваться в каждой команде. Если, например, в микропроцессоре вместо косвенной адресации по регистру используется адресация с индексированием, то команда ADI (СЛОЖЕНИЕ АККУМУЛЯТОРА С ОБЛАСТЬЮ ПАМЯТИ ПРИ КОСВЕННОЙ АДРЕСАЦИИ ПО РЕГИСТРУ) замещается командой ADD A,X (СЛОЖЕНИЕ С ИНДЕКСИРОВАНИЕМ). Команда ADD A,X занимает в памяти 2 байт и выполняется за пять циклов. Обычно для выполнения команд при использовании адресации с индексированием требуется большее число микроциклов, чем для выполнения аналогичных команд с использованием косвенной адресации по регистру.

На рис. 13.3 представлена простая про-

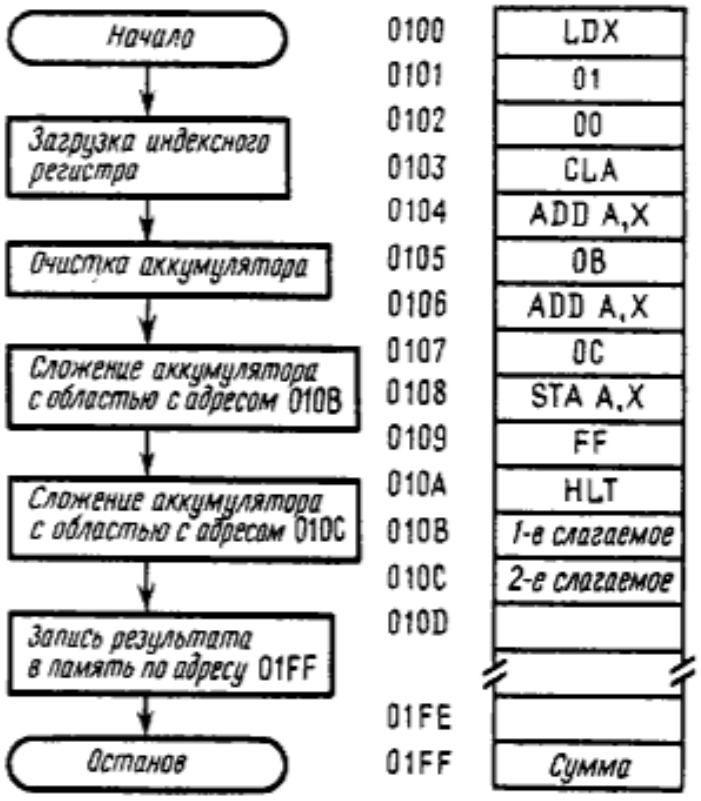


Рис. 13.3. Программа сложения двух чисел, в которой использована адресация с индексированием.

грамма, в которой адресация с индексированием использована в двух командах – ADD A,X и STA A,X (ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ ПРИ АДРЕСАЦИИ С ИНДЕКСИРОВАНИЕМ).

Первая команда рассматриваемой программы LDX обеспечивает загрузку индексного регистра. Прежде чем использовать индексный регистр для адресации, необходимо записать в него определенное начальное значение. В противном случае не будет известно, какая величина содержится в индексном регистре. В данном примере после начальной установки содержимое индексного регистра равно 0100_{16}.

С помощью команды CLA осуществляется сброс содержимого аккумулятора, установка в 0 разрядов переноса и отрицательного результата, а также установка в 1 разряда нулевого результата.

Первой командой, в которой используется индексирование, является команда сложения ADD A,X 0B. Выполняя эту команду, микропроцессор прибавит к содержимому аккумулятора значение области с адресом 010B.

Микропроцессор прибавляет к содержимому аккумулятора именно значение области 010B, поскольку, согласно способу адресации с индексированием, адрес операнда определяется посредством сложения содержимого индексного регистра со вторым байтом команды. Результат такого сложения и является адресом области памяти. В рассматриваемом примере этот адрес вычисляется следующим образом

```
  0100   Содержимое индексного регистра
+   0B   Значение второго байта команды
------
  010B   Адрес области памяти
```

Следует отметить, что вычисленный таким образом адрес области памяти записывается только в регистр адреса памяти; при этом содержимое индексного регистра и величина смещения в выполняемой команде остаются прежними.

В следующей команде программы также используется индексирование. Это команда сложения, в которой (в отличие от предыдущей команды) смещение равно 0C. Следовательно, во время выполнения рассматриваемой команды к содержимому аккумулятора будет прибавлено значение области с адресом 010C.

Теперь в аккумуляторе находится сумма значений областей с адресами 010C и 010B.

Последняя команда, в которой используется способ адресации с индексированием, обеспечивает запись содержимого аккумулятора в область памяти с адресом 01FF. Это команда STA A,X FF. При ее реализации смещение FF складывается с содержимым индексного регистра, которое равно 0100. Выполнение нашей программы завершается командой HLT (ОСТАНОВ). Следует заметить, что, используя адресацию с индексированием, можно написать и другие варианты рассматриваемой программы. Например, можно использовать в командах смещение, равное 00. Кроме того, зачастую адресацию желаемых областей памяти можно обеспечить путем применения команд увеличения и уменьшения содержимого индексного регистра.

Интересно провести сравнение адресации с индексированием и косвенной адресации по регистру. Каждый из этих способов адресации обладает как достоинствами, так и недостатками. Например, команды, в которых используется адресация с индексированием,

занимают в памяти 2 байт, а при использовании косвенной адресации по регистру для каждой команды требуется только по 1 байт. В то же время, применяя две команды с индексированием, легко адресовать два файла памяти, начальные адреса которых отличаются не более чем на 256. В случае косвенной адресации по регистру следует использовать два различных указателя памяти, роль которых могут играть либо другие регистры, либо специально для этого выделенные области памяти. С учетом команд, необходимых для определения значений указателя, при использовании косвенной адресации требуется больше команд, чем при использовании адресации с индексированием.

В некоторых микропроцессорах используется как косвенная адресация, так и адресация с индексированием. Оказывается, что для некоторых задач программы, написанные с использованием отмеченных здесь способов адресации, имеют почти одинаковую длину. Однако иногда небольшие программы можно написать более компактно с помощью какого-либо одного из указанных способов адресации.

Приемы программирования при использовании адресации с индексированием отличаются от приемов программирования в случае косвенной адресации по регистру. Алгоритмы решения задач, ориентированные на реализацию рассматриваемых способов адресации, также имеют некоторые отличия.

На рис. 13.4–13.7 представлены блок-схемы и программы решения одной и той же задачи при использовании косвенной адресации по регистру и адресации с индексированием. Для данной задачи более простой получается программа с применением второго способа адресации. Задача состоит в пересылке массива байтов: исходный массив, расположенный в областях памяти с адресами 004F–006E, пересылается в области с адресами 00CA – 00E9.

На рис. 13.4. представлена блок-схема программы и схема распределения памяти, соответствующие применению косвенной адресации по регистру. Начало исходного и формируемого массивов находится соответственно в областях с адресами 004F и 00CA.

Области с адресами 001A и 001B используются для записи младших байтов указателей, применяемых для адресации областей исходного и формируемого массивов соответственно. Очевидно, что старший байт всех адресов областей исходного и формируемого массивов имеет значение 00. Для загрузки регистра C используем команду LDD C. Для адресации областей исходного массива, адреса которых равны 004F – 006E, в младший байт регистровой пары BC записывается содержимое области 001A; аналогично для адресации областей 00CA – 00E9 формируемого массива в младший байт регистровой пары BC записывается содержимое области 001B. Таким образом, регистровая пара BC поочередно используется для адресации областей исходного и формируемого массивов.

Первая команда программы предназначена для загрузки в старший байт регистровой пары BC значения 00. Следующая команда LDD C,001A обеспечивает загрузку регистра C, после которой регистровая пара BC будет содержать адрес 004F. При следующем обращении к области с адресом 001A ее содержимое должно обеспечить адресацию следующего элемента исходного массива. Требуемое изменение содержимого области 001A можно осуществить, если переслать содержимое регистра C в аккумулятор, увеличить содержимое регистра A (аккумулятора) на 1 и поместить содержимое аккумулятора в область с адресом 001A. Выполнение указанных операций, очевидно, обеспечивает такое изменение содержимого области с адресом 001A, при котором будет адресоваться следующая область исходного массива.

Теперь можно осуществить пересылку данных. Для этого используем команду с косвенной адресацией. Напомним, что выполненная ранее пересылка содержимого регистра C в аккумулятор не изменила содержимого регистра C. Следовательно, содержимое регистровой пары BC равно адресу первой области исходного массива.

Теперь используется команда загрузки аккумулятора с применением косвенной адресации; в данном случае эта команда предназначается для записи содержимого первой области исходного массива в аккумулятор.

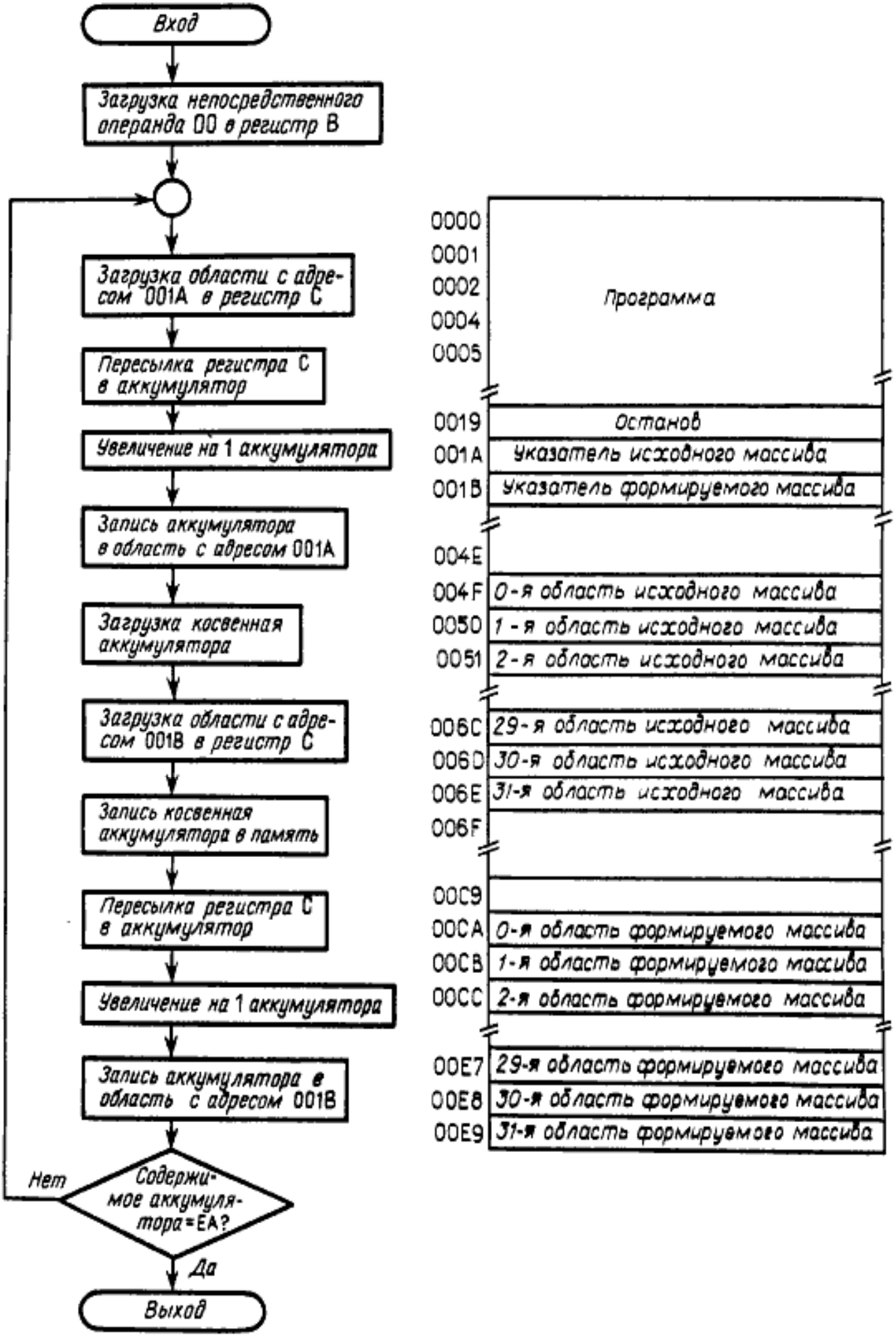


Рис. 13.4. Блок-схема программы и схема распределения памяти, ориентированные на использование микропроцессора с косвенной адресацией по регистру для выполнения операции пересылки массива данных.

Адрес области памяти	Команда	Комментарий
0000	LDA B	Запись 0 в СБ регистровой пары BC
0001	00	
0002	LDD C	Загрузка в регистр C указателя исходного массива
0003	00	
0004	1A	
0005	MOV A,C	Пересылка содержимого регистра C в аккумулятор
0006	INC A	Увеличение на 1 указателя исходного массива
0007	STA A	Запись в память указателя исходного массива
0008	00	
0009	1A	
000A	LDI A	Загрузка в аккумулятор элемента исходного массива
000B	LDD C	Загрузка в регистр C указателя формируемого массива
000C	00	
000D	1B	
000E	STI A	Запись содержимого аккумулятора в формируемый массив
000F	MOV A,C	Пересылка содержимого регистра C в аккумулятор
0010	INC A	Увеличение на 1 указателя формируемого массива
0011	STA A	Запись в память указателя формируемого массива
0012	00	
0013	1B	
0014	CMP	Массив сформирован?
0015	EA	
0016	JNZ	Нет, продолжение пересылки элементов массива

Рис. 13.5. Программа, в которой используется косвенная адресация для пересылки массива данных.

Адрес области памяти	Команда	Комментарий
0017	00	
0018	02	
0019	HLT	Да, останов
001A	4F	Начальное значение указателя исходного массива
001B	CA	Начальное значение указателя формируемого массива

Рис. 13.5. Продолжение.

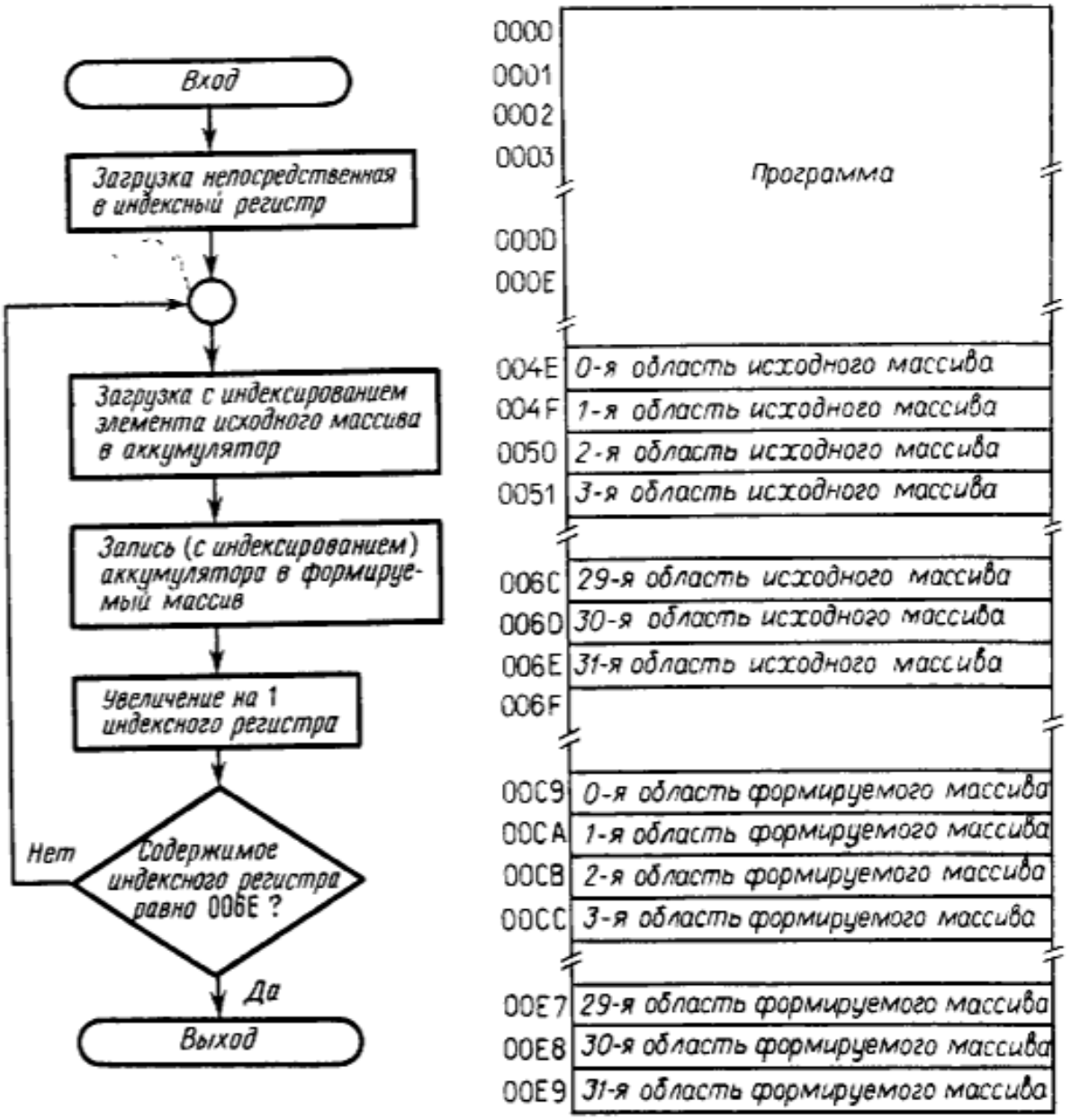


Рис. 13.6. Блок-схема программы и схема распределения памяти, ориентированные на использование микропроцессора с адресацией с индексированием, для выполнения операции пересылки массива данных.

Для пересылки содержимого области памяти с адресом 001B в регистр С используется команда загрузки в регистр С содержимого области памяти с указанным в команде адресом. Таким образом, проведена подготовка для пересылки данных в организуемый массив. Содержимым регистровой пары является адрес 00CA, который представляет собой адрес первой области организуемого массива. Для записи содержимого аккумулятора в первую область организуемого массива можно воспользоваться командой ЗАПИСЬ АККУМУЛЯТОРА В ПАМЯТЬ ПРИ КОСВЕННОЙ АДРЕСАЦИИ ПО РЕГИСТРУ.

При очередной записи значения в организуемый массив регистровая пара BC должна содержать адрес следующей области массива. Подобно тому как выполнялось изменение содержимого области с адресом 001A, в данном случае осуществляется пересылка содержимого регистра С в аккумулятор, увеличение содержимого аккумулятора на 1 и запись обновленного значения в область памяти с адресом 001B.

Итак, 1 байт данных переслан из исходного массива в организуемый массив. Эта операция должна быть повторена еще 31раз. После выполнения последней пересылки данных и увеличения содержимого аккумулятора на 1 в нем окажется код EA. С помощью команды сравнения производится проверка содержимого аккумулятора на равенство коду EA. Если в аккумуляторе действительно содержится код EA, то при выполнении команды сравнения разряд нулевого результата в регистре состояния будет установлен в состояние 1. В зависимости от значения упомянутого разряда команда перехода, расположенная в областях памяти с адресами 0016–0018, либо обеспечит переход по программе назад (в этом случае будет продолжена пересылка данных), либо приведет к завершению работы программы, т.е. выполнению команды ОСТАНОВ.

Теперь рассмотрим, каким образом можно выполнить действия по пересылке данных из одного массива в другой, применяя адресацию с индексированием. На рис. 13.6 и 13.7 представлены блок-схема, схема распределения памяти и текст программы, при реализации которой использована адресация с индексированием. Первая команда программы предназначена для записи в индексный регистр адреса первой области исходного массива.

С помощью команды LDA A,X00 в аккумулятор загружается содержимое области с адресом 004F. Смещение в этой команде равно нулю, поскольку в индексном регистре уже содержится необходимый адрес области памяти.

Следующая команда STA A,X7B обеспечивает запись содержимого аккумулятора в область памяти по адресу 00CA. Адрес 00CA, по которому производится запись содержимого аккумулятора, вычисляется посредством сложения смещения 7B и содержимого индексного регистра, равного 004F. Следует отметить, что величина смещения 7B (123_{10}) фактически равна разности адресов первых областей в исходном и формируемом массивах.

Для увеличения содержимого индексного регистра на 1 используется команда INX. Теперь индексный регистр позволит нам адресовать следующую область исходного массива. Благодаря изменению содержимого индексного регистра во время выполнения программы при фиксированных смещениях 00 и 7B обеспечивается адресация нужных областей исходного и формируемого массивов.

После того как содержимое индексного регистра будет 32 раза увеличено на 1, в нем окажется величина, равная 006E. Можно считать, что содержимое индексного регистра при этом адресует 33-ю область исходного массива. После каждого увеличения содержимого индексного регистра с помощью команды CPX производится его сравнение с величиной 006E. Если при выполнении команды сравнения CPX разряд нулевого результата принимает значение 1 (сравниваемые величины равны), то выполнение программы прекращается, в противном случае, т.е. когда значение разряда нулевого результата равно 0, команда JNZ обеспечивает переход по программе назад для осуществления пересылки очередного байта данных.

Используя адресацию с индексированием, нам удалось написать весьма простую программу для решения частной задачи пересылки данных. Однако если разность между адресами первых областей исходного

Адрес области памяти	Команда	Комментарий
0000	LDX	Загрузка в индексный регистр указателя исходного массива
0001	00	
0002	4F	
0003	LDA A,X	Загрузка в аккумулятор элемента исходного массива
0004	00	Нулевое смещение
0005	STA A,X	Запись содержимого аккумулятора в формируемый массив
0006	7B	Формируемый массив отстоит от исходного на 7B
0007	INX	Увеличение на 1 содержимого индексного регистра
0008	CPX	Массив сформирован?
0009	00	
000A	6E	
000B	JNZ	Нет, продолжение пересылки элементов массива
000C	00	
000D	03	
000E	HLT	Да, останов

Рис. 13.7. Программа, в которой используется адресация с индексированием для пересылки массива данных.

и формируемого массивов окажется больше чем 256, то из-за ограничения на величину смещения рассмотренный метод не может быть применен. Если разность начальных адресов исходного и формируемого массивов превышает 256, то при использовании адресации с индексированием, как и в случае применения косвенной адресации по регистру, потребуется два 16-разрядных указателя. Один указатель соответствует исходному массиву, другой – формируемому массиву.

Задания для самопроверки

1. Допустим, что в программе, представленной на рис. 13.3, в область с адресом 0100 вместо команды LDX 0100 записана команда LDX 010B. Какие еще изменения необходимо сделать в программе, чтобы по результатам работы она была эквивалентна исходной программе? Приведите текст модифицированной программы.

2. Каким образом в программе, представленной на рис. 13.3, можно вместо первой команды сложения ADD A,X воспользоваться командой LDA A,X (ЗАГРУЗКА АККУМУЛЯТОРА С ИНДЕКСИРОВАНИЕМ)? Какие еще команды должны быть изменены? Приведите новый текст программы.

3. Напишите для рассматриваемого здесь гипотетического микропроцессора вариант программы, представленной на рис. 13.3, без использования при этом индексного регистра. Какой вариант программы является более эффективным? Почему?

4. Ниже приведены команды, в которых используется адресация с индексированием. Каким образом будут выполняться эти команды, если в индексном регистре находятся следующие величины:

Содержимое индексного регистра	Команда
а) 01F0	ADD A,X 00
б) 0005	STA A,X 05
в) 0011	LDA A,X 00
г) 00FF	INX
д) A1AE	SUB A,X AE
е) A105	AND A,X A0
ж) FFFF	DEX
з) 0AEE	CLA

5. Каким образом должна быть изменена программа, представленная на рис. 13.4, чтобы с ее помощью можно было бы пересылать массивы любой длины?

6. На рис. 13.7 представлена программа пересылки 32-байтового массива из одного места памяти в другое. а) Какова максимальная длина массива, пересылку которого можно выполнить с помощью данной программы? б) Чему равна наибольшая величина разности между адресами первых областей в исходном и формируемом массивах? Дайте обоснование ваших ответов.

7. Почему в программе, представленной на рис. 13.7, в области памяти 000B записана команда JNZ 0003, а не команда JNZ 0000?

8. Чем отличается выполнение приведенной ниже программы от программы, представленной на рис. 13.7?

```
0000    LDX
0001    00
0002    6E
0003    LDA A,X
0004    00
0005    STA A,X
0006    7B
0007    DEX
0008    CPX
0009    00
000A    4F
000B    JNZ
000C    00
000D    03
000E    HLT
```

13.2. Относительная адресация

В каждой команде некоторых микропроцессоров, главным образом 16-разрядных, может быть использована *относительная адресация*. Известны микропроцессоры, в которых данный способ адресации может быть применен только в нескольких командах.

Относительная адресация имеет много общего с адресацией с индексированием. Однако эти способы адресации имеют и отличия. Во-первых, при использовании рассматриваемых способов адресации смещение складывается с содержимым разных регистров; при относительной адресации смещение прибавляется к текущему содержимому счетчика команд. Во-вторых, в режиме относительной адресации применяется специальная форма арифметических операций над числами, представленными в дополнительном коде. Применение такой «специальной арифметики» позволяет выполнять переходы вперед в тех случаях, когда в старшем разряде смещения содержится значение, равное логическому 0. Кроме того, оказывается возможным производить переходы назад, когда значение старшего разряда смещения равно 1. Смещение при этом рассматривается как число, представленное в дополнительном коде.

Почему же иногда вместо адресации с индексированием используют относительную адресацию? В наборе команд некоторых микропроцессоров имеются команды, в которых предусматривается использование

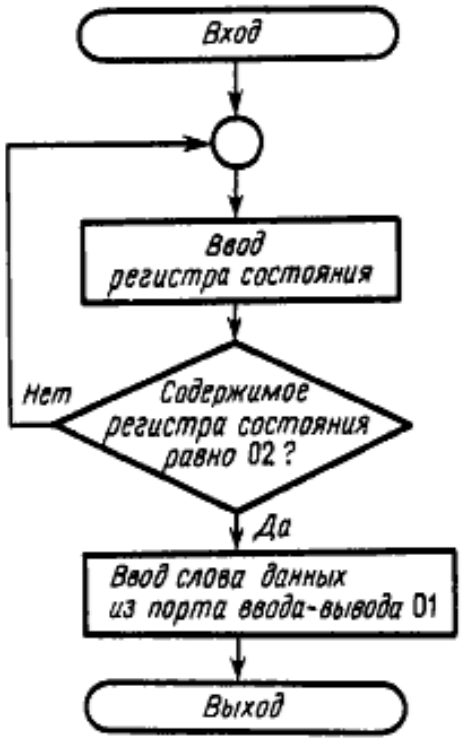


Рис. 13.8. Блок-схема алгоритма анализа значения флажка «Данные готовы» в регистре состояния входного порта канала последовательной передачи данных.

только относительной адресации. Примером такой команды может служить команда условного перехода. Кроме того, относительная адресация позволяет разрабатывать *перемещаемые* программы. Такие программы называют также *относительными программами* или *позиционно-независимыми программами*, т. е. программами, не зависимыми от местоположения. При относительной адресации в команде записывается величина разности между адресом требуемой области и содержимым счетчика команд, соответствующим адресу подлежащей выполнению команды. Следовательно, одна и та же программа может быть размещена в различных областях памяти.

На рис. 13.8 представлена блок-схема алгоритма, который может быть запрограммирован с использованием как абсолютной, так и относительной адресации, т. е. этот алгоритм может быть реализован в виде абсолютной или перемещаемой программы. «Абсолютная программа» представляет собой программу, местоположение которой в памяти постоянно. На рис. 13.9, *а* представлена абсолютная программа, а на рис. 13.9, *б* – относительная, или перемещаемая, программа.

Рассматриваемая программа предназначена для ввода данных из регистра состояния универсального асинхронного приемопередатчика. С помощью команды ANI – И НЕПОСРЕДСТВЕННОЕ анализируется значение флажка «Данные готовы» в регистре состояния универсального приемопередатчика (этому флажку соответствует первый разряд упомянутого регистра). Если флажок «Данные готовы» установлен в состояние 1, то в аккумуляторе после выполнения указанной команды окажется результат, отличный от 0. В этом случае будет выполняться следующая команда, т. е. команда ввода слова данных из порта ввода-вывода 01.

Если в регистре состояния приемопередатчика разряд «Данные готовы» имеет значение 0, то после выполнения команды ANI во всех разрядах аккумулятора будут находиться значения, равные 0. В этом случае произойдет переход по команде JZ (ПЕРЕХОД, ЕСЛИ НУЛЬ). Как раз в этой точке и различаются рассматриваемые программы. В программе, представленной на рис. 13.9, *а*, производится передача управления по адресу 0000. В программе, изображенной на рис. 13.9, *б*, в команде условного перехода используется относительная адресация. Здесь передается управление назад к команде, адрес которой отличается на 6 от текущего значения счетчика команд. Напомним, что в то время, когда выполняется команда условного перехода в нашей программе, содержимое счетчика команд равно адресу команды ввода IN, следующей за командой условного перехода. По команде перехода микропроцессор произведет переход назад на число «шагов», равное числу, представленному в дополнительном коде во втором байте команды перехода. В рассматриваемом случае число шагов при переходе будет равно шести, поскольку величина FA представляет собой дополнительный код числа 6. В самом деле, команда JZ RFA указывает микропроцессору на необходимость вычитания из содержимого счетчика команд числа 6. Дополнительный код числа 6 определяется следующим образом:

0000 0110	Число 6 в двоичном коде
1111 1001	Обратный код числа 6
0000 0001	Число 1 в двоичном коде
1111 1010	Число 6 в дополнительном коде
F A	Шестнадцатеричное смещение

Адрес области памяти	Команда	Комментарий
0000	IN	Ввод содержимого регистра состояния
0001	00	
0002	ANI A	Установлен ли флажок
0003	02	«Данные готовы»?
0004	JZ	Нет, переход к вводу содержимого регистра состояния
0005	00	
0006	00	
0007	IN	Да, ввод слова данных из порта 01
0008	01	
0009	RET	Выход из программы

а

Адрес области памяти	Команда	Комментарий
0000	IN	Ввод содержимого регистра состояния
0001	00	
0002	ANI A	Установлен ли флажок
0003	02	«Данные готовы»?
0004	JZR	Нет, переход назад на 6 байт к команде
0005	FA	ввода содержимого регистра состояния
0006	IN	Да, ввод слова данных
0007	01	
0008	RET	Выход из программы

б

Рис. 13.9. *а*) Программа ввода данных, в которой использована команда условного перехода с прямой адресацией, и *б*) программа ввода данных, содержащая команду условного перехода с относительной адресацией.

Адрес области памяти	Команда	Комментарий
01AE	IN	Ввод содержимого регистра состояния
01AF	00	
01B0	ANI A	Установлен ли флажок
01B1	02	«Данные готовы»?
01B2	JZ R	Нет, переход назад на 6 байт к команде
01B3	FA	ввода содержимого регистра состояния
01B4	IN	Да, ввод слова данных
01B5	01	
01B6	RET	Выход из программы

Рис. 13.10. Программа ввода данных, аналогичная программе, представленной на рис. 13.9, *б*, и отличающаяся от нее только местом расположения в памяти. (Настоящая программа расположена в памяти, начиная с области 01AE.)

Очевидно, что программы, представленные на рис. 13.9, *а*, *б*, занимают в памяти примерно одинаковое количество областей. Однако программа, в которой используется относительная адресация, на 1 байт короче программы, реализованной с использованием абсолютной адресации. Это объясняется тем, что в программе, представленной на рис. 13.9, *а*, команда условного перехода, имеющая адрес 0004, занимает в памяти 3 байт.

На рис. 13.10 представлена такая же программа, что и на рис. 13.9, *б*. Но в данном случае адрес первой области программы равен 01AE. Какие-либо отличия в мнемонических обозначениях, кодах операций или в адресных частях рассматриваемых программ отсутствуют. Передача управления на шесть областей назад по отношению к текущему содержимому счетчика команд приводит теперь к области с адресом 01AE, являющимся адресом команды ввода содержимого регистра состояния; производится переход к требуемой команде несмотря на то, что содержимое счетчика команд равно теперь 01B4.

Перемещение программы, представленной на рис. 13.9, *а*, в другую область памяти вызвало бы внесение некоторых изменений в программу. Так, вместо команды JZ 0000 следовало бы записать команду JZ 01AE.

Очевидно, что при использовании относительной адресации программист не должен беспокоиться относительно того, где в памяти машины будет размещена его программа. Однако он должен хорошо знать шестнадцатеричную (или восьмеричную) арифметику, используемую при вычислении относительного адреса, хотя использование ассемблера и освобождает программиста от вычисления такого адреса: необходимые вычисления выполняет ассемблер.

Задания для самопроверки

9. Почему в программе, представленной на рис. 13.9, *а*, вместо 3-байтовой команды условного перехода, в которой использована прямая адресация (область с адресом 0004), не применена 2-байтовая команда условного перехода с индексированием?

10. Команда, в которой используется относительная адресация, всегда занимает в памяти 2 байт. Чему равен диапазон адресации по отношению к адресу команды в памяти?

11. Поясните ход выполнения следующих фрагментов программ, в которых использованы команды с относительной адресацией:

```
а) 010F  LDA  A        б) 00DE  JMP  R
   0110   JZ  R           00DF   01
   0111   F1             00E0  CLA  A
   0112  RET             00E1  CLA  B
в) 0021  INX           г) 0FAB  ADD  B
   0022  JNZ  R           0FAC  JNC  R
   0023   0D             0FAD   8D
   0024  DEX              0FAE  RET
```

12. Часто в микропроцессорах пространство памяти размером в 256 областей, для адресации которых может быть использовано 1-байтовое смещение, называют «страницей». Почему, применяя способ относительной адресации, можно говорить о «плавающей странице»?

13.3. Команды работы со стеком

В этом разделе мы будем изучать команды, позволяющие работать со стеком микропроцессора. Нам известно, что стек микропроцессора применяется либо при выполнении команд вызова подпрограмм, либо при обработке прерываний. Операции со стеком при этом выполняются автоматически, т.е. они не задаются явно программистом.

Команды, рассматриваемые в настоящем разделе, предназначены для выполнения операций со стеком, не имеющих отношения к вызову подпрограмм и обработке прерываний. Программист, использующий для работы стек, должен быть очень внимательным. Загрузка данных в стек и извлечение их из стека должны четко контролироваться. Если загрузить в стек некоторые данные и не извлечь их оттуда перед реализацией другой команды микропроцессора, выполнение которой связано с некоторыми действиями со стеком, то возможны катастрофические последствия для работы программы.

Напомним, что стек представляет собой устройство памяти, действующее по принципу LIFO – last in, first out (поступивший последним обрабатывается первым). Это означает, что данные, загруженные в стек последними, будут извлекаться из стека первыми. Извлечение данных из стека производится в порядке, обратном загрузке в стек.

Предположим, что программируется обращение к двум подпрограммам, одна из которых вызывает другую. Вызываемую подпрограмму будем называть подпрограммой второго уровня. На рис. 13.11 показано содержимое стека во время выполнения подпрограммы второго уровня. Байт стека со старшим адресом занимает область памяти с адресом 0F25. В качестве начальной области стека была выбрана произвольная область памяти, которая не должна использоваться в других частях программы.

Содержимое областей с адресами 0F25 и 0F24 равно адресу области главной программы. Этот адрес является адресом команды, следующей в главной программе за командой вызова подпрограммы. Области памяти с адресами 0F25 и 0F24 содержат соответственно младший и старший байты счетчика команд. Когда завершится выполнение подпрограммы, работа главной программы будет продолжена, начиная с

Адрес области памяти	Программа	Стек
0F21	Следующий доступный элемент стека	
0F22	Старший байт СК для первой подпрограммы	02
0F23	Младший байт СК для первой подпрограммы	01
0F24	Старший байт СК для главной программы	00
0F25	Младший байт СК для главной программы	0C

Рис. 13.11. Состояние стека микропроцессора во время выполнения подпрограммы второго уровня

команды, расположенной в области памяти с адресом 000С. Отметим, что при выборке данных из стека сначала будет извлечен старший, а затем младший байт счетчика команд.

Из рисунка также можно видеть, что за командой вызова подпрограммы второго уровня в подпрограмме первого уровня следует команда с адресом 0201.

В рассматриваемом примере подпрограмма второго уровня обеспечивает умножение чисел с тройной точностью. Во время выполнения этой подпрограммы необходимо выделить временную область памяти, используемую при выполнении команды сдвига влево содержимого 3-байтового поля. На рис. 13.12 показано размещение каждого байта множимого, занимающего 3 байт памяти в стеке. На рис. 13.12, *а* изображено состояние стека после пересылки в него младшего байта множимого. Указатель стека, изображенный на рисунке стрелкой, определяет здесь, как и везде, следующую свободную область стека. Состояние стека после загрузки в него второго байта множимого изображено на рис. 13.12, *б*, а после загрузки последнего (старшего) байта – на рис. 13.12, *в*.

На рис. 13.12, *г – е* показано состояние стека после выборки данных из стека. Стек, изображенный на рис. 13.12, *е*, уже не содержит множимого. Оставшаяся в стеке информация – это два значения счетчика команд, которые были показаны на рис. 13.11. Следует отметить, что множимое не содержится в стеке, поскольку изменилось значение указателя стека; содержимое же областей с адресами 0F1E – 0F21 в действительности будет сохраняться прежним до тех пор, пока при загрузке новых данных в стек не будет осуществлена запись по указанным адресам.

Рассмотренный нами пример позволяет сделать вывод о важности соблюдения как порядка, так и количества выполнений операций загрузки данных в стек и операций извлечения данных из стека. Предположим, что старший байт множимого по ошибке не был извлечен из стека. Это могло произойти, если вызов подпрограммы умножения был произведен, когда стек имел состояние, подобное изображенному на рис. 13.12, *а*, а ее завершение по каким-то причинам произошло, когда состояние стека было подобно состоянию, представленному на рис. 13.12, *д*. Тогда по завершении подпрограммы умножения из стека извлекаются 2 байт и помещаются в счетчик команд. Затем микропроцессор будет выполнять подпрограмму первого уровня. В рассматриваемом примере из стека извлекаются 2 байт, значение которых дает адрес области памяти, равный 7202. Однако начальный адрес подпрограммы первого уровня равен 0201. Таким образом, при вызове подпрограммы будет использован неправильный адрес, что, скорее всего, приведет к нарушению функционирования микропроцессорной системы.

Ранее указывалось, что большинство микропроцессоров имеет по несколько команд, с помощью которых программист может производить операции со стеком.

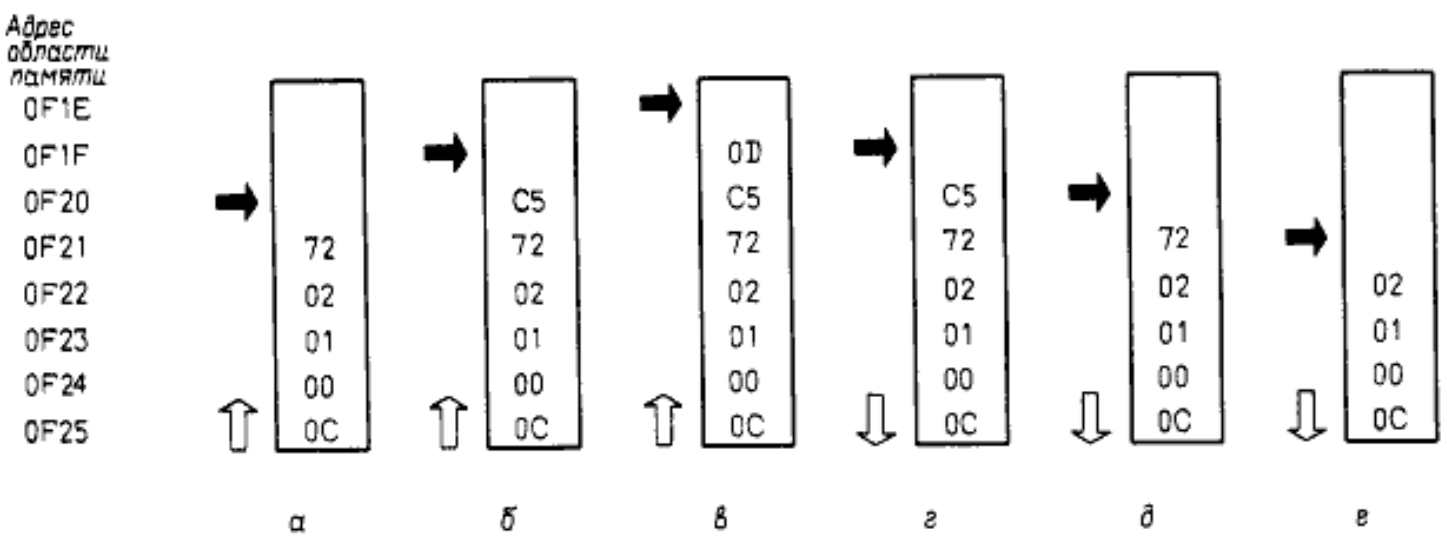
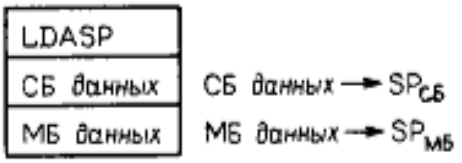


Рис. 13.12. Добавление данных в стек, используемый для временного хранения данных при сдвиге содержимого 3-байтового поля.

В рассматриваемом нами микропроцессоре предусмотрены следующие команды работы со стеком:

ЗАГРУЗКА НЕПОСРЕДСТВЕННАЯ В УКАЗАТЕЛЬ СТЕКА

LDA SP,Данные

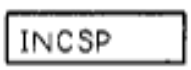


Команда загрузки непосредственного операнда в указатель стека занимает в памяти 3 байт и выполняется за три цикла. Эта команда используется для установки исходного состояния стека, что обеспечивается загрузкой в указатель стека адреса начальной области стека. Напомним, что очередная доступная для использования область стека имеет меньший адрес по отношению к адресам других, занятых областей стека. Следовательно, команда загрузки непосредственного операнда в указатель стека будет использоваться для задания области стека с наибольшим адресом. Второй байт команды определяет старший байт указателя стека. Младший байт указателя стека задается третьим байтом команды. При выполнении этой команды в регистре состояния будет устанавливаться в 1 разряд нулевого результата, если 16-разрядное загружаемое слово имеет значение 0, и разряд отрицательного результата, если старший разряд загружаемого в указатель стека слова имеет значение 1. Команда загрузки в указатель стека непосредственного операнда выполняется один раз перед началом работы со стеком.

Стек никогда не устанавливается сам в исходное состояние! Следовательно, каждый раз перед вызовом подпрограммы или обработкой прерывания в программе программист должен использовать команду LDA SP для задания указателя стека.

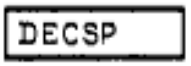
ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ УКАЗАТЕЛЯ СТЕКА

INC SP SP + 1 → SP



ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ УКАЗАТЕЛЯ СТЕКА

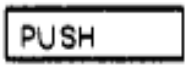
DEC SP SP − 1 → SP



Команды увеличения и уменьшения значения указателя стека выполняются над содержимым 16-разрядного регистра. При выполнении этих команд содержимое регистра состояния не изменяется.

ЗАГРУЗКА АККУМУЛЯТОРА В СТЕК

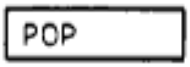
PUSH ACC → Стек
SP − 1 → SP



По этой команде содержимое аккумулятора загружается в стек, а содержимое указателя стека уменьшается на 1. Значение разрядов регистра состояния при выполнении этой операции не изменяется.

ЗАГРУЗКА СТЕКА В АККУМУЛЯТОР

POP SP + 1 → SP
Стек → аккумулятор



По этой команде производятся извлечение данных из стека и загрузка их в аккумулятор. Пересылке подлежит байт данных, который был записан в стек последним. Содержимое указателя стека увеличивается на 1. Содержимое регистра состояния при выполнении этой команды не изменяется.

Задания для самопроверки

13. Почему непосредственно после выполнения команды загрузки в стек не требуется использовать команду уменьшения на 1 содержимого указателя стека?

14. Команда загрузки содержимого аккумулятора в стек использована для временного сохранения в стеке слова данных, которое должно потребоваться после выполнения нескольких шагов программы. Если в дальнейшем окажется, что необходимость

в извлечении указанного слова данных из стека с помощью команды POP отсутствует, то какую команду следует использовать для предотвращения возможных грубых нарушений работы микропроцессорной системы?

15. Для разработки программы обеспечения пересылки данных из одного массива в другой, при условии что разность между адресами первых областей массивов превышает величину FF, требуется использовать два индексных регистра. Программирование такой передачи данных может быть осуществлено и с применением для индексирования указателя стека. Дайте объяснение возможности реализации второго подхода для составления программы пересылки данных.

16. Какой недостаток свойствен подходу, предложенному в предыдущем упражнении? Какие условия необходимо соблюдать при написании вышеупомянутой программы для того, чтобы обеспечить ее корректное выполнение?

Упражнения

13.1. При использовании адресации с индексированием адрес области памяти определяется как сумма текущего содержимого индексного регистра и содержащегося в команде а) абсолютного адреса, б) косвенного адреса, в) смещения или г) непосредственного операнда?

13.2. С помощью какой из указанных ниже команд выполняется установка начального значения индексного регистра?

а) СБРОС НЕПОСРЕДСТВЕННЫЙ
б) ЗАГРУЗКА ИНДЕКСНОГО РЕГИСТРА
в) СБРОС ИНДЕКСНОГО РЕГИСТРА
г) ПРИРАЩЕНИЕ ОТРИЦАТЕЛЬНОЕ ИНДЕКСНОГО РЕГИСТРА

13.3. Какой из приведенных здесь объектов представляет собой индексный регистр, используемый в большинстве 8-разрядных микропроцессоров:

а) регистровая пара с определенными старшим и младшим байтами;

б) регистр адреса памяти;

в) 1-байтовая команда, выполняемая за два цикла;

г) 16-разрядный регистр?

13.4. Команда CPX применяется: а) для сброса индексного регистра, б) для установки начального состояния индексного регистра, в) для сравнения содержимого индексного регистра с заданной величиной или г) для сложения смещения с содержимым индексного регистра?

13.5. Смещение, прибавляемое к содержимому индексного регистра, обычно принимает значение, лежащее в диапазоне от 0 до а) 255_{10}, б) FF_{16}, в) 377_8 или г) во всех указанных здесь диапазонах?

13.6. Используются два различных смещения. Объясните, как повлияет на эти смещения увеличение или уменьшение на 1 содержимого индексного регистра.

13.7. Укажите, в какое устройство [а) аккумулятор, б) индексный регистр, в) регистр адреса памяти или г) счетчик команд] помещается результат сложения содержимого индексного регистра и смещения.

13.8. Какой из указанных здесь способов адресации обычно не применяется в 8-разрядных микропроцессорах, если одновременно используется индексная адресация:

а) относительная адресация; б) прямая адресация; в) непосредственная адресация; г) косвенная адресация по регистру?

13.9. Объясните, чем отличаются команды ADD A,X и ADI.

13.10. Какая область памяти будет адресоваться при выполнении следующих ниже команд?

Содержимое индексного регистра	Команда	
а) 0000	JP	X 0F
б) 0100	ADD	A,X 00
в) 0100	ADD	A,X FF
г) 1000	ADD	A,X 00
д) 1016	ADD	A,X FA
е) FFFF	ADD	A,X 00
ж) FFFF	ADD	A,X FA

13.11. Сравните адресацию с индексированием, относительную и косвенную по регистру. Что общего между ними?

13.12. Чему равна при использовании адресации с индексированием предельная величина разности между адресами начальных областей исходного и формируемого массивов при пересылке массива данных?

13.13. Команда перехода, в которой используется относительная адресация, имеет смещение, равное 8B. На сколько байт удалена от команды перехода область памяти с адресом 008F, в которую осуществляется переход? В каком направлении (вперед или назад) будет осуществлен этот переход в программе? Дайте обоснование ответам.

13.14. Выполнение команды условного перехода с относительной адресацией зависит от со-

стояния двух регистров. Назовите эти регистры. Каким образом выполнение команды перехода зависит от содержимого этих регистров?

13.15. Каким преимуществом обладает перемещаемая программа?

13.16. Укажите серьезную опасность, которой может подвергаться программа при использовании стека для хранения данных.

13.17. Для передачи данных из стека в аккумулятор используется команда: а) MOV A, б) POP, в) PUSH или г) STA A?

13.18. Начальная установка указателя стека производится:

а) с помощью команды, помещающей в указатель стека адрес области памяти, имеющий наибольшее значение среди всех адресов областей, выделенных для организации стека;

б) посредством использования команды LDX;

в) при выполнении команды LRP B или

г) путем выполнения команды LSP?

13.19. Что произошло бы, если бы сигнал прерывания поступил сразу после включения питания, но перед выполнением начальной установки указателя стека?

13.20. Какие меры предосторожности следует принять в подпрограмме, предназначенной для пересылки массива с использованием указателя стека для индексирования?

Ответы на вопросы заданий для самопроверки

1. В начале программы содержимое индексного регистра указывает на первое слагаемое. Следовательно, первая команда сложения ADD A,X имеет смещение, равное 0. Следующая команда сложения может иметь смещение, равное либо 01, либо 00. Если используется смещение, равное 00, то перед выполнением второй команды сложения содержимое индексного регистра должно быть увеличено на 1. Ниже приведены тексты вариантов рассматриваемой программы.

```
0100        LDX  или      LDX
0101          01            01
0102          0B            0C
0103        CLA           CLA
0104        ADD A,X       ADD A,X
0105          00            00
0106        ADD A,X       INX
0107          01          ADD A,X
0108        STA A,X         00
0109          F4          STA A,X
010A        HLT             F3
010B Первое слагаемое     HLT
010C Второе слагаемое     Первое слагаемое
010D                      Второе слагаемое
```

2. Используя команду LDA, получим программу, представленную ниже. В этот вариант программы не включена команда CLA.

```
0100        LDX
0101          01
0102          00
0103        LDA A,X
0104          0A
0105        ADD A,X
0106          0B
0107        STA A,X
0108          FF
0109        HLT
010A      Первое слагаемое
010B      Второе слагаемое
```

3. Обе программы – новая, текст которой приведен ниже, и программа, представленная на рис. 13.3, занимают одинаковое количество областей памяти.

```
0100        LRP B
0101        00
0102        0B
0103        CLA
0104        ADI
0105        INC C
0106        ADI
0107        STA A
0108        00
0109        FF
010A        HLT
010B      Первое слагаемое
010C      Второе слагаемое
01FF
```

4. а) Содержимое области памяти 01F0 прибавляется к содержимому аккумулятора;

б) содержимое аккумулятора пересылается в область памяти по адресу 000A;

в) содержимое области памяти 0011 загружается в аккумулятор;

г) содержимое индексного регистра теперь равно 0100;

д) содержимое области памяти с адресом A25C вычитается из содержимого аккумулятора;

е) производится операция И над содержимым области памяти с адресом A1A5 и содержимым аккумулятора;

ж) содержимое индексного регистра становится равным FFFE;

з) осуществляется очистка аккумулятора; эта команда выполняется без индексирования.

5. Потребуется применить два 16-разрядных указателя – по одному для каждого массива. Эти указатели должны храниться в памяти и переда-

ваться в индексный регистр или в регистровую пару BC каждый раз, когда возникает необходимость в адресации областей соответствующего массива. После использования для индексирования значения указателей должны увеличиваться на 1 и помещаться в память в выделенные для них области.

6. а) Максимальная длина массива равна 255 байт.

б) Максимальное значение разности адресов начальных областей массивов равно 255 (FF_{16})

Указанные ограничения обусловлены предельным значением смещения, равным FF.

7. Благодаря этому не требуется каждый раз выполнять перезагрузку индексного регистра. Целесообразно, чтобы в этой программе значение индексного регистра изменялось только командой INC, имеющей адрес 0007

8. В этой программе сначала пересылается содержимое области массива, имеющей наибольший адрес. Содержимое индексного регистра после пересылки очередного элемента массива уменьшается на 1. Таким образом пересылаются все элементы массива. В программе, представленной на рис. 13.7, пересылка ведется начиная с области, имеющей в массиве наименьший адрес.

9. В данной программе индексирование вообще не используется. Поскольку значение индексного регистра либо не определено, либо задано в других частях программы, его использование в данном случае нецелесообразно.

10. Используя представление чисел в дополнительном коде, можно в 1-байтовой области хранить числа диапазона от -128 до $+127$. Поскольку текущее значение счетчика команд равно адресу области, расположенной за выполняемой командой, а последняя в нашем случае имеет длину 2 байт, диапазон адресации по отношению к адресу выполняемой команды определяется границами -126, $+129$.

11. а) Если при выполнении команды LDA A в аккумулятор загружается число 0, то по команде условного перехода будет выполняться переход назад по адресу 0103. В противном случае произойдет переход к следующей команде (RET).

б) Выполняется очистка регистра B. Команда CLA A не будет выполняться.

в) Если после выполнения команды увеличения на 1 содержимого индексного регистра его содержимое оказывается не равным 0, то по команде условного перехода будет произведен переход к области с адресом 0031. Если же после увеличения на 1 содержимого индексного регистра его содержимое оказывается равным 0, то будет выполнена команда уменьшения на 1 содержимого индексного регистра.

г) Если при выполнении команды сложения перенос не происходит, то будет осуществлен переход по адресу 103D; в противном случае выполнится команда RET.

12. Адресуемую область памяти в диапазоне от $+127$ до -128 относительно содержимого счетчика команд можно условно называть плавающей страницей, поскольку при выполнении команд программы содержимое счетчика команд претерпевает изменение.

13. После выполнения команды загрузки в стек не требуется использовать команду уменьшения на 1 содержимого указателя стека, поскольку это делается автоматически.

14. В таких случаях необходимо использовать команду INC SP (ПРИРАЩЕНИЕ ПОЛОЖИТЕЛЬНОЕ УКАЗАТЕЛЯ СТЕКА), с помощью которой будет восстановлено требуемое для нормального выполнения программы содержимое указателя стека.

15. Для индексирования двух массивов можно использовать один указатель стека, однако для этого придется выделить память для сохранения двух значений указателя стека, соответствующих индексируемым массивам; эти значения придется должным образом корректировать и при обращении к элементам массивов поочередно загружать обновленные значения в указатель стека.

16. Недостатком предложенного в п. 15 приема использования стека для индексирования массивов при пересылке данных является то, что при этом стек не может служить для других целей. Следовательно, если массивы данных нельзя использовать в качестве временных стеков, необходимо быть уверенным, что стек в программе не будет применяться для каких-либо целей, в частности не будут производиться обращения к подпрограммам и не будут происходить прерывания.

Глава 14. Аппаратные средства микропроцессорных систем

В настоящей главе рассматриваются аппаратные средства, применяемые в микропроцессорных системах. До настоящего момента основное внимание было уделено архитектуре микропроцессора и используемому набору команд. Однако следует помнить, что область применения микропроцессора в значительной мере определяется используемыми совместно с ним аппаратными средствами.

Вначале микропроцессор рассматривается как часть аппаратных средств микропроцессорных систем. Дается представление о схеме генератора тактовых импульсов, сигналах, передаваемых по шине микропроцессорной системы, и сигналах управления, необходимых для обеспечения передачи данных. Далее описывается несколько интегральных схем, которые используются для расширения возможностей применения микропроцессора. После изучения этого материала читатель сможет самостоятельно разобраться в стандартных способах организации интерфейса и адресации. Кроме того, будет рассмотрена схема таймера и показано, каким образом он используется в системе прерываний.

В заключение речь пойдет об однокристальном микропроцессоре, что облегчит знакомство с основными характеристиками аналогичных устройств и условиями их применения. Наконец, будут обсуждены аппаратные средства, используемые при разработке микропроцессорных систем и их [illegible]ции. Читателю станет понятна необходимость в создании систем разработки микропроцессоров. Будет дано представление о методике использования логического анализатора, логического пробника и сигнатурного анализатора.

14.1. Микропроцессор как техническое устройство

Большинство микропроцессоров имеют корпусы с 40 или 64 выводами, расположенными в два ряда. Значительное количество выводов микропроцессора выделяется для ввода-вывода данных, передачи адреса, управляющих сигналов и подключения питающего напряжения. Рассмотрим некоторые типичные сигналы микропроцессора.

Большая часть выводов микропроцессора используется для подключения линий шины данных и шины адреса. Обычно 8-разрядный микропроцессор имеет восемь линий данных, обозначаемых D0–D7 и 16 адресных линий A0–A15. Для указанных линий необходимо выделить 24 вывода из 40 выводов микропроцессора. В некоторых микропроцессорах оставшихся 16 выводов оказывается недостаточно. Однако фирмы-изготовители микропроцессоров стремятся избегать применения относительно дорогих 64-выводных корпусов.

Чтобы сократить число выводов, используемых для подключения линий данных и адресных линий, применяют способ, называемый *мультиплексированием*. Обратимся к рис. 14.1. Очевидно, что ли-

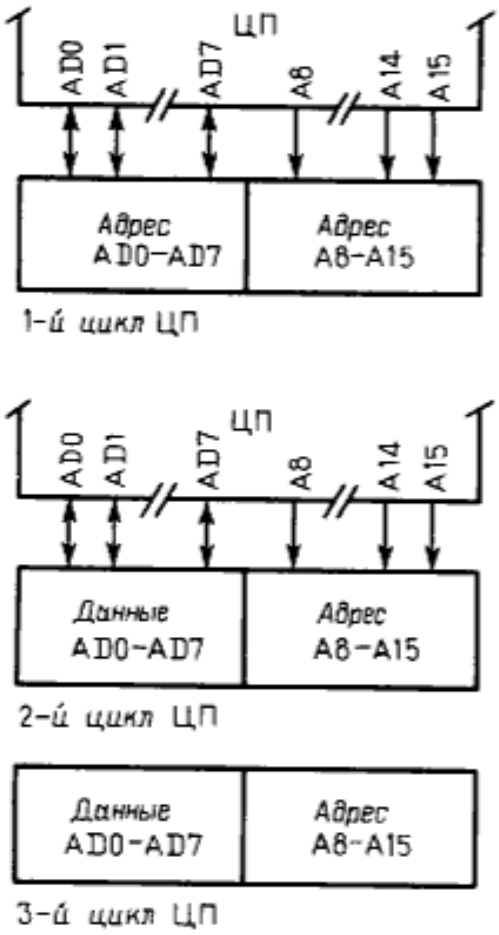


Рис. 14.1. Распределение времени занятости мультиплексированной шины для передачи информации разного типа.

нии AD0–AD7 используются как для передачи адреса памяти, так и для передачи данных. Эти восемь линий во время первого периода следования тактовых импульсов микропроцессора служат для передачи адреса памяти. В течение 2-го и 3-го периодов следования тактовых импульсов указанные линии используются как линии двунаправленной передачи данных. Адресные линии A8–A15 используются только для передачи адреса памяти.

Благодаря мультиплексированию удается передавать по 16-разрядной шине и адреса памяти, и данные. Чтобы обеспечивать обмен информацией микропроцессора с устройствами по мультиплексированной шине данных, в микропроцессоре должны быть предусмотрены специальные выходы. Следовательно, несколько выводов корпуса микропроцессора должны использоваться для выдачи информации о состоянии микропроцессора и синхронизирующих сигналов.

Микропроцессор с мультиплексированной шиной данных должен, например, в то время когда на линиях AD0–A15 установятся уровни, соответствующие определенному адресу, подать сигнал «Строб адреса». Этот сигнал используется для обеспечения фиксации адреса в определенном внешнем, или периферийном, устройстве. После передачи адресной информации в некоторое периферийное устройство микропроцессор выдает либо сигнал чтения (записи) из памяти (в память), либо сигнал чтения-записи для внешнего устройства ввода-вывода. Такой сигнал сообщает периферийному устройству, что на линиях AD0–AD7 уже установились уровни, соответствующие передаваемым данным. Фактически эти сигналы указывают как тип устройства – память или устройство ввода-вывода, участвующий в операции, так и вид подлежащей выполнению операции – ввод или вывод данных.

Некоторые микропроцессоры кроме выхода для сигнала «Строб адреса» имеют несколько выходов для выдачи сигналов о состоянии ЦП. Дешифрирование этих сигналов позволяет определить, в каком состоянии – выборка команды, выполнение команды, чтение из памяти, запись в память, чтение из устройства ввода, запись в устройство вывода, останов – находится микропроцессор.

Адресные линии и линии данных микропроцессора обычно рассчитывают на нагрузку, определенную для ТТЛ-элементов. На входах шины данных также обеспечивается совместимость с элементами ТТЛ-логики, т.е. на входы шины данных могут подаваться сигналы того же уровня, что и на ТТЛ-элементы.

Для синхронизации микропроцессора должна быть предусмотрена подача последовательности тактовых импульсов, или синхроимпульсов. Тактовые импульсы используются для синхронизации всех операций обработки информации и передачи данных. Все операции в микропроцессоре происходят синхронно с сигналами, вырабатываемыми генератором тактовых импульсов.

Для обеспечения работы некоторых микропроцессоров требуется подача двухфазной последовательности тактовых импульсов. Сигналы первой и второй фаз ($\phi 1$ и $\phi 2$), как показано на рис. 14.2, не перекрываются. Исходной для образования

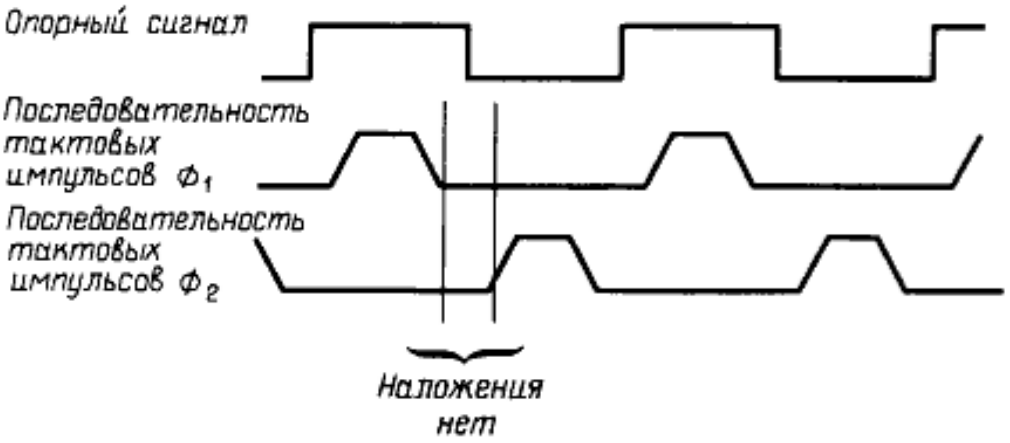


Рис. 14.2. Двухфазная последовательность тактовых импульсов.

сигналов ф1 и ф2 является одна основная последовательность импульсов. Одному периоду исходной последовательности сигналов соответствуют четыре временные точки в последовательностях формируемых сигналов: две точки определяют передний фронт сигналов ф1 и ф2, а две – задний фронт этих сигналов. Следует отметить, что амплитудные характеристики рассматриваемых сигналов часто не совпадают со стандартом, принятым для ТТЛ-схем. Эти сигналы обычно изменяются от уровня «Земля» до $E_{пит}$, а диапазон изменения уровня типичных сигналов ТТЛ-схем имеет границы 0,4 и 2,4 В.

На практике для генерирования последовательности тактовых импульсов используются два различных подхода. Первый подход заключается в изготовлении генераторов тактовых импульсов в виде специальных интегральных схем. Такие схемы наряду со схемой генератора содержат элементы, предназначенные для формирования сигналов ф1 и ф2. Кроме того, они часто используются для генерирования сигнала начальной установки микропроцессора, т. е. сигнала «Сброс». Может быть предусмотрено формирование и других синхронизирующих сигналов.

При втором подходе генератор тактовых импульсов «встраивается» в схему микропроцессора. Однако для установления определенной частоты следования генерируемых тактовых импульсов используется внешняя схема, в качестве которой обычно применяется кварцевый резонатор. Все необходимые сигналы синхронизации в данном случае вырабатываются микропроцессором.

На рис. 14.3 представлены четыре ти-

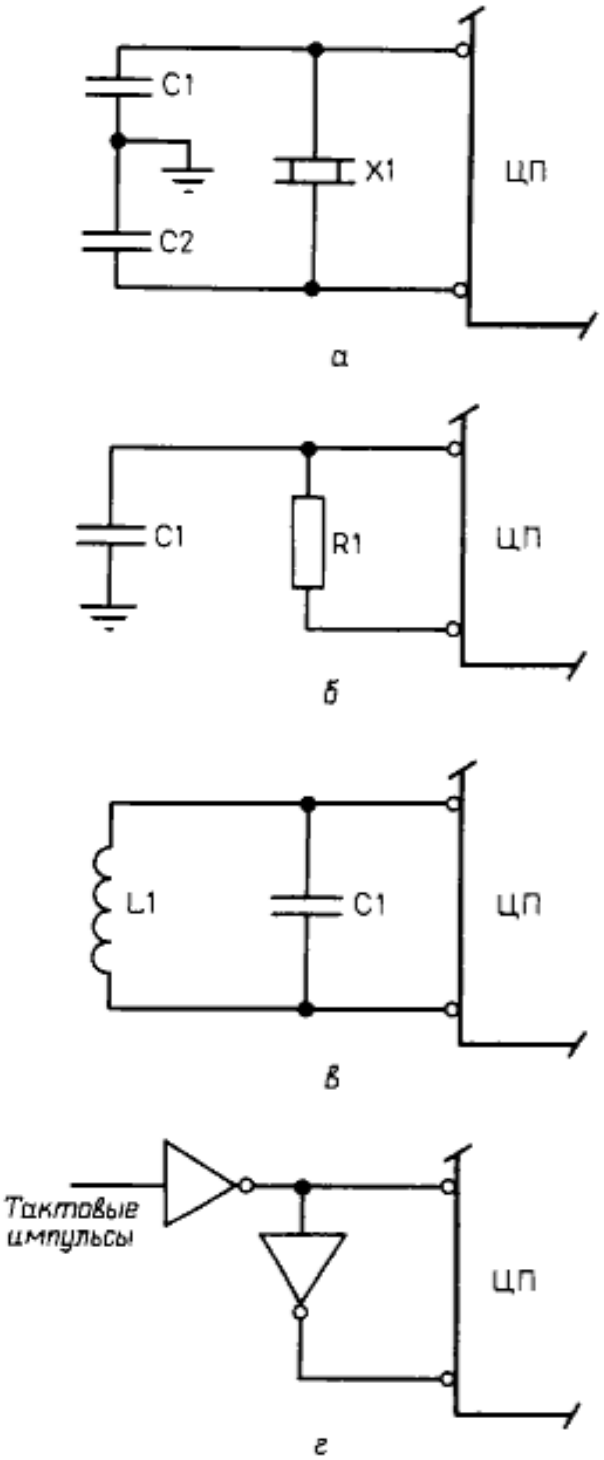


Рис. 14.3. Четыре способа задания опорной частоты генератора тактовых импульсов: а) с помощью кварцевого резонатора; б) с помощью RC-генератора; в) с помощью LC-генератора; г) с помощью внешнего источника сигнала.

пичные схемы, используемые для задания частоты следования тактовых импульсов. Обычно частота импульсов, формируемых с помощью рассматриваемых схем, в 2–4 раза превышает частоту следования тактовых импульсов микропроцессора. Посредством деления исходной частоты, выполняемого с помощью делителей частоты, получают требуемые последовательности тактовых импульсов.

На рис. 14.3, *а* представлена наиболее часто используемая схема с кварцевым резонатором, которая обеспечивает выработку опорной частоты генератора тактовых импульсов. Обычно для стабилизации частоты генераторов тактовых импульсов микропроцессоров применяют кварцевые резонаторы с частотой колебаний 1–20 МГц. На рис. 14.3, *б* и 14.3, *в* представлены более дешевые внешние дополнительные элементы генератора тактовых импульсов, задающие опорную частоту колебаний. Показанный на рис. 14.3, *б* RC-генератор отличается низкой стабильностью частоты. Генератор с LC-контуром (рис. 14.3, *в*) обладает несколько лучшей характеристикой стабильности частоты генерируемых сигналов. Ни одна из этих двух схем не может быть использована на частоте, превышающей 5 МГц. Следовательно, обе эти схемы можно использовать для микропроцессоров низкого быстродействия, допускающих невысокую точность синхронизации сигналов.

Иногда требуется синхронизировать микропроцессор посредством сигналов от некоторого внешнего источника. На рис. 14.3, *г* представлена схема, с помощью которой внешний источник синхросигналов подключается к микропроцессору.

Микропроцессоры, имеющие встроенные генераторы тактовых импульсов, имеют и выходы для выдачи синхронизирующих сигналов. Эти сигналы могут использоваться для синхронизации устройств микропроцессорной системы, которые должны работать синхронно с микропроцессором. Обычно на определенный выход микропроцессора подается вторая последовательность синхроимпульсов, т.е. последовательность ф2.

Для подачи напряжения питания служат по крайней мере два входа; число входов, задействованных с этой целью, иногда достигает четырех. В современных микропроцессорах используется, как правило, одинаковое напряжение питания. Обычно источником питания для микропроцессора является источник постоянного тока напряжением + 5 В. Для подключения этого источника используются два вывода микропроцессора: на один из них подается напряжение + 5 В, второй вывод заземляется. В некоторых микропроцессорах предусматриваются еще два вывода, предназначенные для подачи напряжений питания + 12 и − 5 В.

Требования по стабильности напряжения и тока источника питания зависят от типа применяемых интегральных схем. Для многих микропроцессоров характерна сравнительно большая потребляемая мощность, обычно составляющая 0,5–1,5 Вт. Исключение составляют микропроцессоры, выполненные на комплементарных МОП-схемах. Такие микропроцессоры отличаются низкой потребляемой мощностью, равной лишь нескольким милливаттам, и высоким быстродействием.

Каждый микропроцессор должен иметь вход «Сброс», предназначенный для подачи сигнала начальной установки микропроцессора. Этот сигнал является своеобразным запросом на прерывание. Он, в частности, используется для обеспечения начальной установки микропроцессора сразу же после включения питания. Большинство входных управляющих сигналов микропроцессора активируется путем заземления входных линий. При этом на вход микропроцессора подается логический сигнал, которому соответствует низкий уровень напряжения. Такой способ представления логических значений сигналов носит название *отрицательной логики*. Вход может заземляться с помощью либо ключа, либо транзистора. Возможные варианты использования с этой целью транзистора представлены на рис. 14.4, *а* и 14.4, *б*.

Сигнал «Сброс» подается с использованием «отрицательной логики» представления его возможных значений. Для формирования этого сигнала часто используются триггер Шмидта и RC-цепочка, показанная на рис. 14.5. Сигнал «Сброс» находится

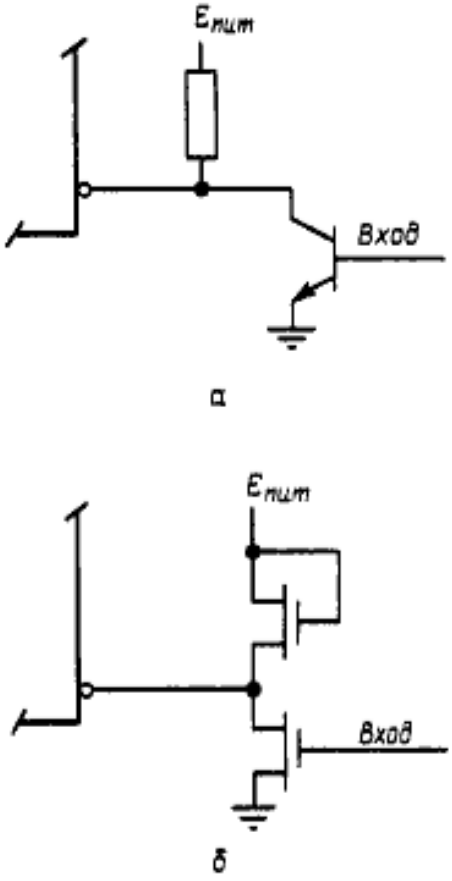


Рис. 14.4. Схемы формирования входных сигналов микропроцессора: а) схема на биполярном транзисторе, рассчитанная на использование ТТЛ-выхода с открытым коллектором; б) схема на МОП-транзисторе.

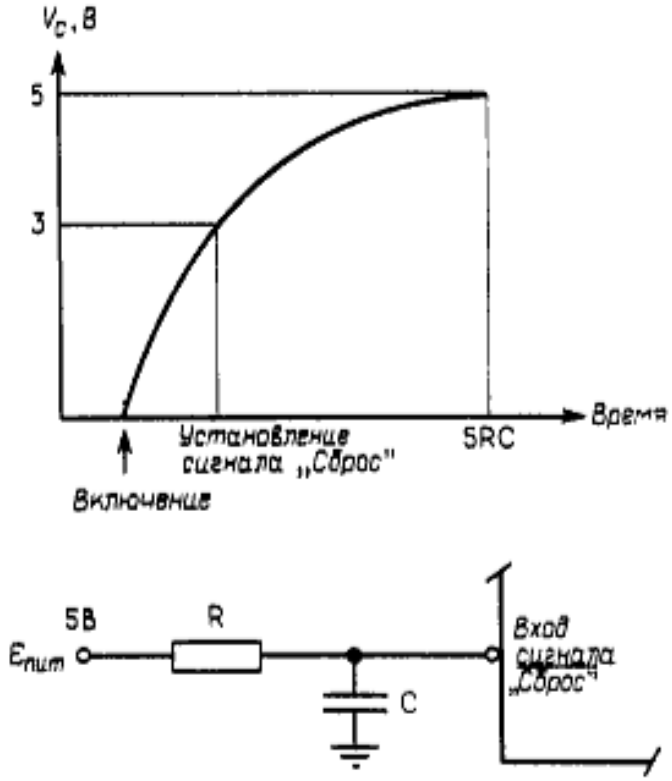


Рис. 14.5. Схема формирования сигнала начальной установки микропроцессора.

Микропроцессор находится в состоянии начальной установки до тех пор, пока конденсатор не зарядится до напряжения, равного 1/2 $E_{пит}$. Продолжительность пребывания в этом состоянии определяется постоянной времени, зависящей от величины сопротивления и емкости RC-цепочки

в активном состоянии, пока конденсатор не зарядится до напряжения +3 В. Выбирая величины сопротивления резистора и емкости конденсатора рассматриваемой схемы, можно поддерживать сигнал «Сброс» в активном состоянии до тех пор, пока не стабилизируется состояние определенных схем микропроцессора.

Если на входе «Сброс» микропроцессора триггера Шмидта нет, то сигнал «Сброс» должен формироваться с помощью внешней схемы, обеспечивающей отсутствие искажений сигнала.

Установка начального состояния микропроцессора состоит в установке начального состояния ряда регистров микропроцессора и некоторых его схем, используемых для управления. При этом в счетчик команд загружается либо значение 0000, либо содержимое двух определенных областей памяти. Эти области памяти называются *вектором начальной установки* микропроцессора.

Микропроцессор, кроме того, может генерировать и выходной сигнал «Сброс». Этот сигнал обычно используется для осуществления начальной установки микропроцессорной системы и формируется специальными аппаратными средствами, входящими в генератор тактовых импульсов. Выходной сигнал «Сброс» снимается в определенный момент времени перед началом первого цикла «Выборка-выполнение».

Большинство микропроцессоров имеет по крайней мере два входа для подачи запросов на прерывания. Один вход обычно используется для запросов на немаскируемые прерывания, другой – для маскируемых прерываний. В микропроцессорах с несколькими входами для маскируемых прерываний устанавливаются приоритеты прерываний, запросы на которые поступают на соответствующие входы. Если одновременно поступают запросы на два прерывания, то сначала воспринимается запрос на прерывание более высокого приоритета. При поступлении запроса на прерывание более высокого приоритета, чем приоритет прерывания, обрабатываемого в данный момент, программа обработки текущего прерывания может быть прервана. Для выполнения программной обра-

ботки прерывания после поступления запроса на прерывание в счетчик команд должен быть помещен адрес начала программы обработки соответствующего прерывания.

В ответ на поступление запроса на прерывание микропроцессор может сформировать определенный выходной сигнал, подтверждающий принятие запроса на прерывание. Сигнал подтверждения прерывания используется для управления периферийными устройствами, например схемой организации прерываний по приоритету.

Если в микропроцессоре предусмотрена возможность работы в режиме прямого доступа к памяти, то микропроцессор должен предоставлять другим устройствам возможность использования адресных линий и линий данных. Таким образом, для реализации режима прямого доступа к памяти требуется использование по крайней мере двух управляющих сигналов. Через специальный вход запрос на прямой доступ к памяти поступает на определенную схему управления микропроцессора. Для подтверждения принятия запроса на организацию прямого доступа к памяти микропроцессор формирует сигнал подтверждения, поступающий на специальный вывод. Сигнал подтверждения генерируется после того, как микропроцессор завершит выполнение всех текущих действий, связанных с использованием адресной шины и шины данных. Сразу после выдачи сигнала подтверждения микропроцессор переводит шину данных и адресную шину в состояние высокого сопротивления. Пребывание выходов в состоянии высокого сопротивления (или в «третьем устойчивом состоянии») означает, что они отключены или «изолированы» и не находятся ни в состоянии логического 0, ни в состоянии логической 1.

Во время режима прямого доступа к памяти микропроцессор не может выполнять никакие команды, реализация которых связана с передачей сигналов по шине данных или по адресной шине. Микропроцессор не будет выполнять операции до тех пор, пока устройство, осуществляющее прямой доступ к памяти, не снимет сигнал Вход ПДП.

На рис. 14.6 представлена конфигурация выводов 8-разрядного микропроцессо-

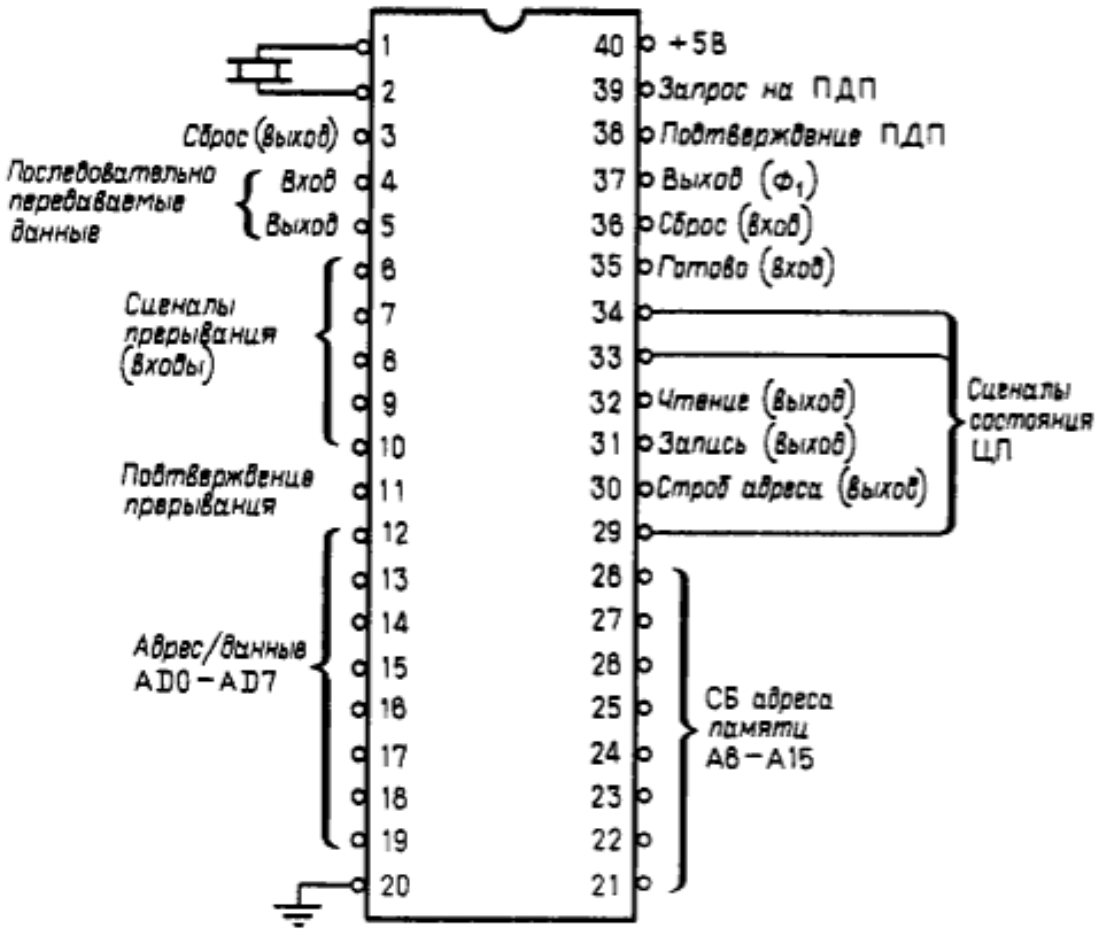


Рис. 14.6. Конфигурация расположения выводов микропроцессора 8085 фирмы Intel.

ра Intel 8085. Многие распространенные в настоящее время микропроцессоры имеют аналогичное распределение выводов. Отметим некоторые особенности данного микропроцессора. Микропроцессор Intel 8085 имеет простой порт последовательного ввода-вывода данных. При выполнении одной команды осуществляется передача только 1 бит данных между седьмым разрядом аккумулятора и некоторым оконечным устройством. Вход «Готово» используется для перевода микропроцессора в состояние ожидания, что связано с работой микропроцессора с устройствами, быстродействие которых ниже его собственного быстродействия. После адресации устройства сигнал «Готово» должен действовать до тех пор, пока устройство не окажется в состоянии готовности для выдачи данных на шину данных микропроцессора.

Назначение других выводов микропроцессора 8085 сходно с назначением выводов рассматриваемого нами микропроцессора.

Задания для самопроверки

1. Почему в микропроцессорах используются как маскируемые, так и немаскируемые входы запросов на прерывания?

2. Большинство управляющих сигналов на входах микропроцессора активируется путем установления на соответствующих входах низкого уровня напряжения. При таком способе поступления сигнала различные источники могут легко переводить в рабочее состояние одну и ту же линию. Почему проще реализовать поступление сигналов посредством установления низкого уровня напряжения, а не высокого? Объясните, почему рассматриваемый способ представления логического значения сигнала называют представлением сигналов с использованием «отрицательной логики».

3. Выполнение какой программы микропроцессора может оказаться не точным, если для задания опорной частоты генератора тактовых импульсов будет использована RC-схема, а не кварцевый резонатор?

4. Какой из нижеперечисленных сигналов не является сигналом управления, вырабатываемым микропроцессором:

а) «Сигнал подтверждения запроса на прямой доступ к памяти»;

б) «Состояние ЦП»;

в) «Чтение устройства ввода-вывода»;

г) «Сброс»?

5. Для какой из указанных ниже целей используется сигнал подтверждения прерывания:

а) для указания, что микропроцессор не может реагировать на запросы на прерывания;

б) для активирования контроллера системы прерываний;

в) для прекращения работы генератора тактовых импульсов микропроцессора;

г) для подачи адресной информации в периферийные устройства микропроцессорной системы с помощью строба?

6. Какие из перечисленных здесь действий необходимо выполнить при использовании для передачи адреса и данных мультиплексированной шины:

а) предусмотреть сигнал стробирования для подачи адресной информации в периферийные устройства;

б) производить передачу данных по линиям данных только в определенное время;

в) подключать адресные линии и линии данных внешних устройств к одним и тем же выходам микропроцессора;

г) выполнить все требования, указанные в предыдущих пунктах?

7. Выдача микропроцессором сигнала подтверждения готовности к работе в режиме прямого доступа к памяти указывает на то, что

а) перестал работать генератор тактовых импульсов,

б) линии данных и адресные линии переведены в состояние высокого сопротивления,

в) произведена начальная установка микропроцессора или

г) осуществлены все перечисленные действия?

8. Сигнал начальной установки микропроцессора

а) используется после подачи питающего напряжения,

б) сопровождается загрузкой в счетчик команд начального адреса программы,

в) обеспечивает подготовку схем управления к выполнению цикла выборки или

г) влечет за собой выполнение всех перечисленных здесь действий?

14.2. Элементы микропроцессорных систем

Для построения микропроцессорных систем было разработано значительное число интегральных схем специального назначения. Некоторые из них, в частности универсальный асинхронный приемопередатчик и другие схемы обеспечения последовательного интерфейса, были рассмотрены выше. Мы также ознакомились с ПЗУ и ОЗУ. Каждое из указанных устройств может быть выполнено в виде специальной интегральной схемы, работающей с определенным микропроцессором или с микропроцессорами некоторого семейства. Эти устройства, кроме того, могут быть реализованы как интегральные схемы общего назначения.

В микропроцессорных системах наряду с другими устройствами широко используются порты параллельного ввода-вывода. Такие порты обычно ориентированы на использование совместно с микропроцессорами некоторого семейства. Порт параллельного ввода-вывода имеет в своем составе схемы адресации, схему для подачи сигнала подтверждения и регистр состояния. Указанные средства обеспечения интерфейса согласуются с управляющими и адресными сигналами, а также с сигналами данных микропроцессора.

Порты параллельного ввода-вывода часто реализуются на одном кристалле вместе с ПЗУ или ПЗУ с возможностью стирания информации. Подобная интегральная схема представлена на рис. 14.7. Эта интегральная схема содержит два двунаправленных 8-разрядных порта ввода-вывода, регистр состояния порта ввода-вывода и ПЗУ емкостью 2048 байт. В качестве постоянного запоминающего устройства могут использоваться различные типы ПЗУ, например ПЗУ с масочным программированием, с плавкими связями или перепрограммируемое ПЗУ.

Приведенная интегральная схема непосредственно связана с адресными линиями микропроцессора, линиями данных и управления. По линиям управления подаются сигналы чтения-записи для устройств ввода-вывода, сигнал чтения памяти, синхронизирующие импульсы и сигнал установки начального состояния микропроцессора. По линии «Сброс» микропроцессор подает сигнал установки в исходное положение регистра состояния, находящегося в рассматриваемой интегральной схеме.

Линии шины данных интегральной схемы соединяются с соответствующими линиями шины данных микропроцессора. По этим линиям данные передаются из памяти, по ним же производится передача вводимых и выводимых данных и управляющей информации. Одиннадцать адресных линий микропроцессора непосредственно соединяются с 11 адресными линиями ПЗУ. Такое количество линий позволяет адресовать

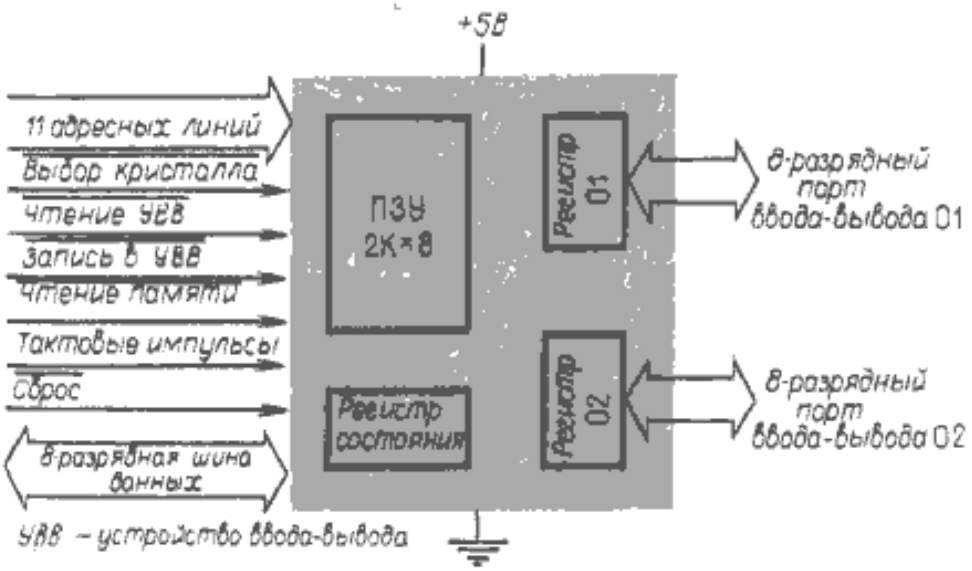


Рис. 14.7. Многофункциональный блок (ПЗУ и средства ввода-вывода), подключаемый непосредственно к адресным линиям, линиям данных и управления.

2048 областей ПЗУ. Эти линии, кроме того, используются для адресации портов ввода-вывода и регистра состояния. Если в системе должно быть более 2048 областей памяти, то выход внешнего дешифратора адреса памяти должен быть подключен к входу ПЗУ «Выбор кристалла». С помощью внешних логических схем и сигнала «Выбор кристалла» обеспечивается выбор нужного модуля ПЗУ.

Еще одним устройством, обычно используемым в микропроцессорных системах, является таймер. Благодаря таймеру можно аппаратным, а не программным путем «генерировать» временны́е задержки. Программный способ установления временны́х задержек был рассмотрен выше, он имеет несколько недостатков. Во-первых, когда выполняется программа, «отсчитывающая время», микропроцессор не может работать по какой-либо другой программе. Он начнет работать по другим программам только по истечении времени задержки. Во-вторых, если требуется обеспечить большое время задержки, то придется использовать при программировании ряд регистров.

Рассматриваемая задача легко решается при использовании программируемого аппаратного таймера. Такой таймер представляет собой счетчик на вычитание, начальное состояние которого может быть установлено программным путем. Когда содержимое счетчика становится равным 0, счетчик вырабатывает импульсный сигнал. Этот сигнал обычно используется как сигнал запроса на прерывание микропроцессора.

У таймера имеется специальный вход для подачи на него тактовых импульсов, поступающих либо от генератора тактовых импульсов микропроцессора, либо от некоторого другого источника синхронизирующих сигналов.

На рис. 14.8 представлено типичное устройство памяти, содержащее таймер. Представленная интегральная схема имеет три порта ввода-вывода, регистр состояния, используемый для управления портами ввода-вывода, 14-разрядный таймер и статическое ОЗУ емкостью 256 байт.

Рассмотренные устройства – микропроцессор, ОЗУ и ПЗУ – позволяют создавать простые микропроцессорные системы на «трех кристаллах». Такая система включает ОЗУ емкостью 256 байт, ПЗУ емкостью 2048 байт, таймер и пять 8-разрядных двунаправленных портов ввода-вывода. При всей своей простоте подобные системы имеют большое практическое значение.

Если таймер, содержащийся в схеме, представленной на рис. 14.8, является 14-разрядным счетчиком на вычитание с программной установкой начального со-

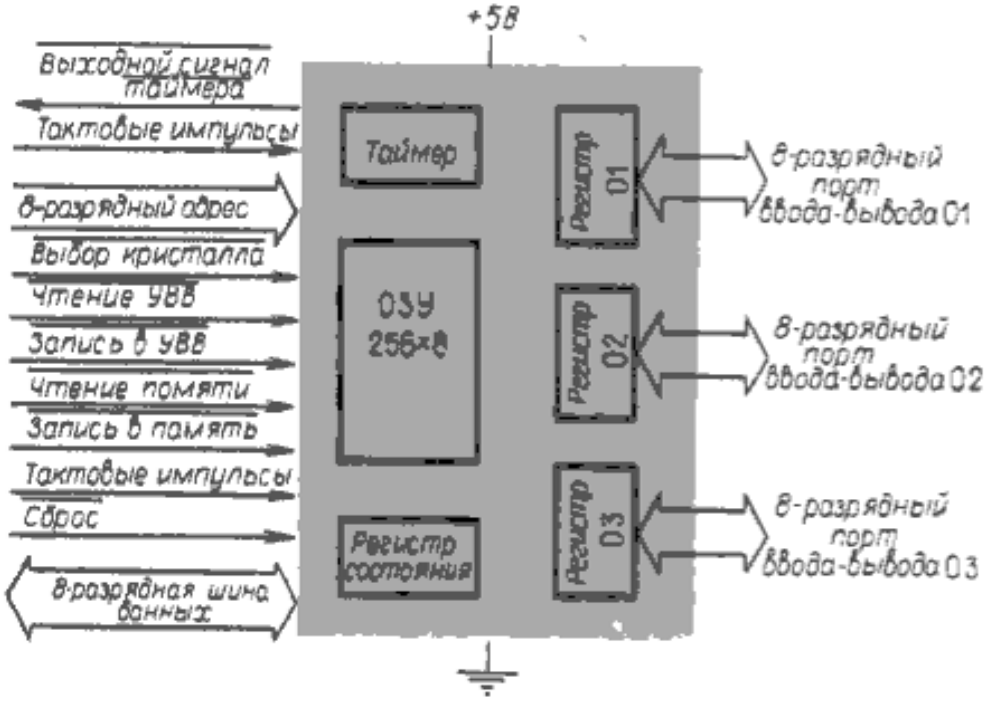


Рис. 14.8. Многофункциональный блок (ОЗУ и средства ввода-вывода)

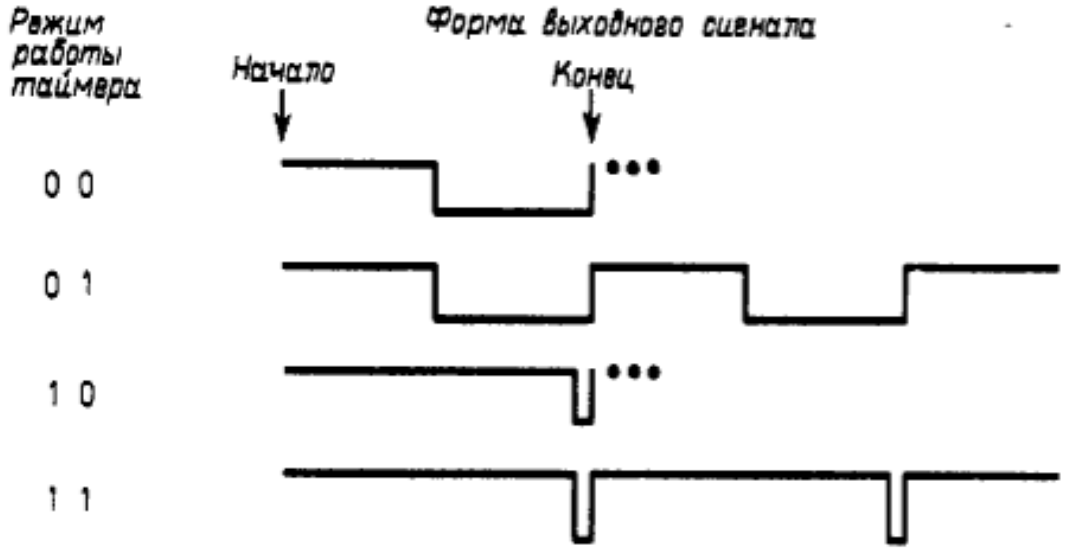


Рис. 14.9. Виды сигналов, формируемых таймером.

стояния, то в него могут быть загружены двоичные числа в диапазоне от 00000000000010 до 11111111111111. Дополнительные 15-й и 16-й разряды таймера используются для управления режимом работы. С возможными режимами работы таймера мы ознакомимся ниже. Используя 14-разрядный таймер, можно задавать временные интервалы, продолжительность которых равна:

Длина интервала времени = Период тактовых импульсов × N,

где N – целое число в диапазоне от 2 до 16 383. С помощью разрядов 15 и 16 можно указать, какой из четырех возможных видов сигнала должен выработать таймер по истечении заданного периода времени. Виды сигналов, которые может формировать таймер, представлены на рис. 14.9. В режимах 01 и 11 таймер вырабатывает периодические последовательности сигналов прямоугольной формы (режим 01) или импульсных сигналов (режим 11). В режимах 00 и 10 таймер генерирует одиночные сигналы. Время пребывания таймера в указанных режимах определяется периодом следования тактовых импульсов. Когда содержимое счетчика достигает значения, равного 0, таймер переходит в состояние ожидания. Запуск таймера осуществляется с помощью специальных команд.

В качестве примера использования таймера могут служить электронные часы или датчик времени ЭВМ. Обычно на вход «Тактовые импульсы» таймера подаются сигналы частотой 60 Гц. Для обеспечения работы датчика времени целесообразно использовать импульсы, следующие с частотой 1 Гц, или один импульс в секунду. Импульсы с желаемым периодом следования можно получать посредством загрузки в счетчик на вычитание величины 3С(60_{10}). При этом после подачи на вход таймера 60 тактовых импульсов он вырабатывает сигнал прерывания микропроцессора.

В рассматриваемом примере таймер вырабатывает сигналы, соответствующие режиму 11 (см. рис. 14.9). Импульсные сигналы на выходе таймера будут появляться через каждую секунду. При появлении этих сигналов происходят прерывания микропроцессора, обработка которых выполняется по программе, осуществляющей подсчет секунд, минут и часов. Блок-схема программы обработки прерывания, вызванного появлением сигнала на выходе таймера, представлена на рис. 14.10. Это типичная программа обеспечения функционирования 12-часового датчика времени. После каждого прерывания по сигналу от таймера эта программа производит необходимые изменения значений счетчиков секунд, минут и часов. Выполнив корректировку значения времени, программа перестает работать вплоть до наступления следующего прерывания.

Для формирования тактовых импульсов, поступающих на вход таймера, обычно используются сигналы одного из трех источников. Во-первых, может быть использован, как и в приведенном выше примере, сигнал частотой 60 Гц, подаваемый по шине питания. Отметим, что в Западной

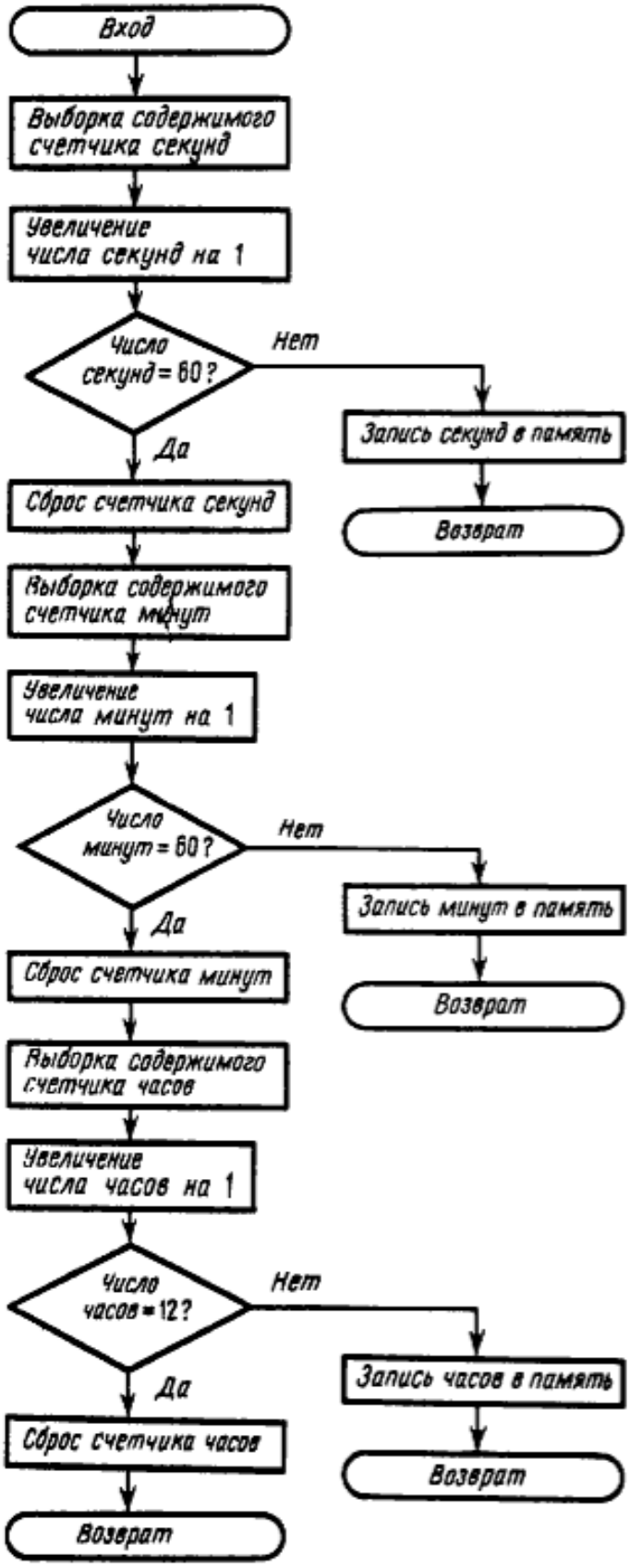


Рис. 14.10. Блок-схема программы датчика времени, обеспечивающей функционирование счетчиков секунд, минут и часов.

Европе напряжение питания имеет частоту 50 Гц[1]; это обстоятельство следует учитывать при разработке программы датчика времени. Во-вторых, часто используются тактовые импульсы, период следования которых равен 1 мс (частота 1 кГц). Для получения колебаний частотой 1 кГц обычно используется опорный сигнал частотой 1 МГц, вырабатываемый генератором с кварцевой стабилизацией частоты. В-третьих, источником тактовых импульсов для таймера может служить генератор тактовых импульсов микропроцессорной системы.

Еще одной интегральной схемой, обычно используемой при проектировании микропроцессорных систем, является двунаправленный шинный *приемопередатчик*, или *формирователь*. Как отмечалось выше, адресная шина и шина данных микропроцессора рассчитываются на использование нагрузки для ТТЛ-схем. Таким образом, к шинам микропроцессора может подключаться 10 и более устройств на комплементарных МОП-схемах, имеющих малую мощность потребления. Однако если шина имеет протяженность, превышающую 0,3 м, и, кроме того, к ней подключены такие ТТЛ-устройства, как дешифраторы, то возникают перегрузки. Для их устранения используют двунаправленные шинные формирователи.

Схема типичного шинного формирователя представлена на рис. 14.11. Такое устройство подключается к четырем линиям шины данных микропроцессора. Линия управления формирователя связана с входными и выходными усилителями. Сигналы, подаваемые по этой линии, обеспечивают управление тристабильной схемой формирователя. Если сигнал «$\overline{\text{Выбор кристалла}}$» активирован, то одна группа усилителей – либо входные, либо выходные – находится в активном состоянии, а другая группа – в состоянии высокого сопротивления.

После подачи сигнала «$\overline{\text{Ввод данных}}$» на соответствующий вход формирователя выходы усилителей 1, 2, 3 и 4 переходят в состояние высокого сопротивления. При этом усилители 5, 6, 7 и 8 находятся в активном состоянии. В таком режиме работы микропроцессор может выполнять передачу данных в определенные устройства мик-

[1] В СССР принят стандарт, в соответствии с которым напряжение питающей сети имеет частоту 50 Гц.– *Прим. перев.*

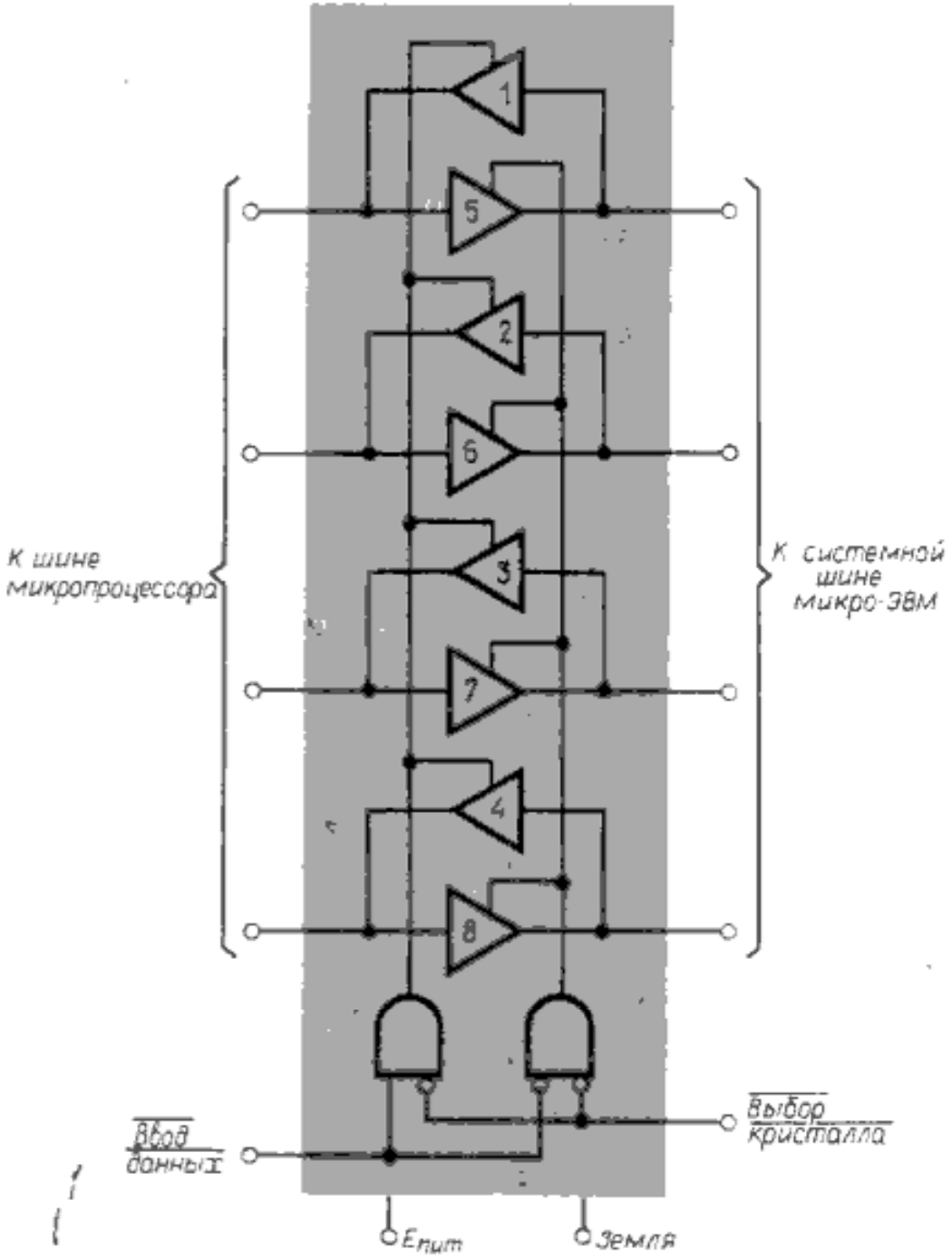


Рис. 14.11. Двунаправленный шинный формирователь.

ропроцессорной системы. Если упомянутый сигнал не активирован, то входы усилителей 5, 6, 7 и 8 находятся в состоянии высокого сопротивления, а посредством сигналов на выходах усилителей 1, 2, 3 и 4 шина данных микропроцессора может управлять системной шиной данных микро-ЭВМ.

Шинные формирователи обладают низкой нагрузочной способностью по входу – они имеют высокое входное сопротивление. Однако благодаря низкому выходному сопротивлению им присуща сравнительно высокая способность по выходу. Указанные особенности формирователей обеспечивают при подключении их к шинам микропроцессора уменьшение постоянной времени, которая в противном случае в значительной мере определяется емкостью самих шин. Емкость любой шины микропроцессора возрастает с увеличением длины шины. Наличие этой емкости ограничивает скорость передачи данных по шине. Шинные формирователи часто используются в микропроцессорных системах в качестве буферов для ЦП. Кроме того, они применяются в интерфейсах памяти и интерфейсах устройств ввода-вывода, выполняя в них роль буферов. На рис. 14.12 представлена типичная микро-

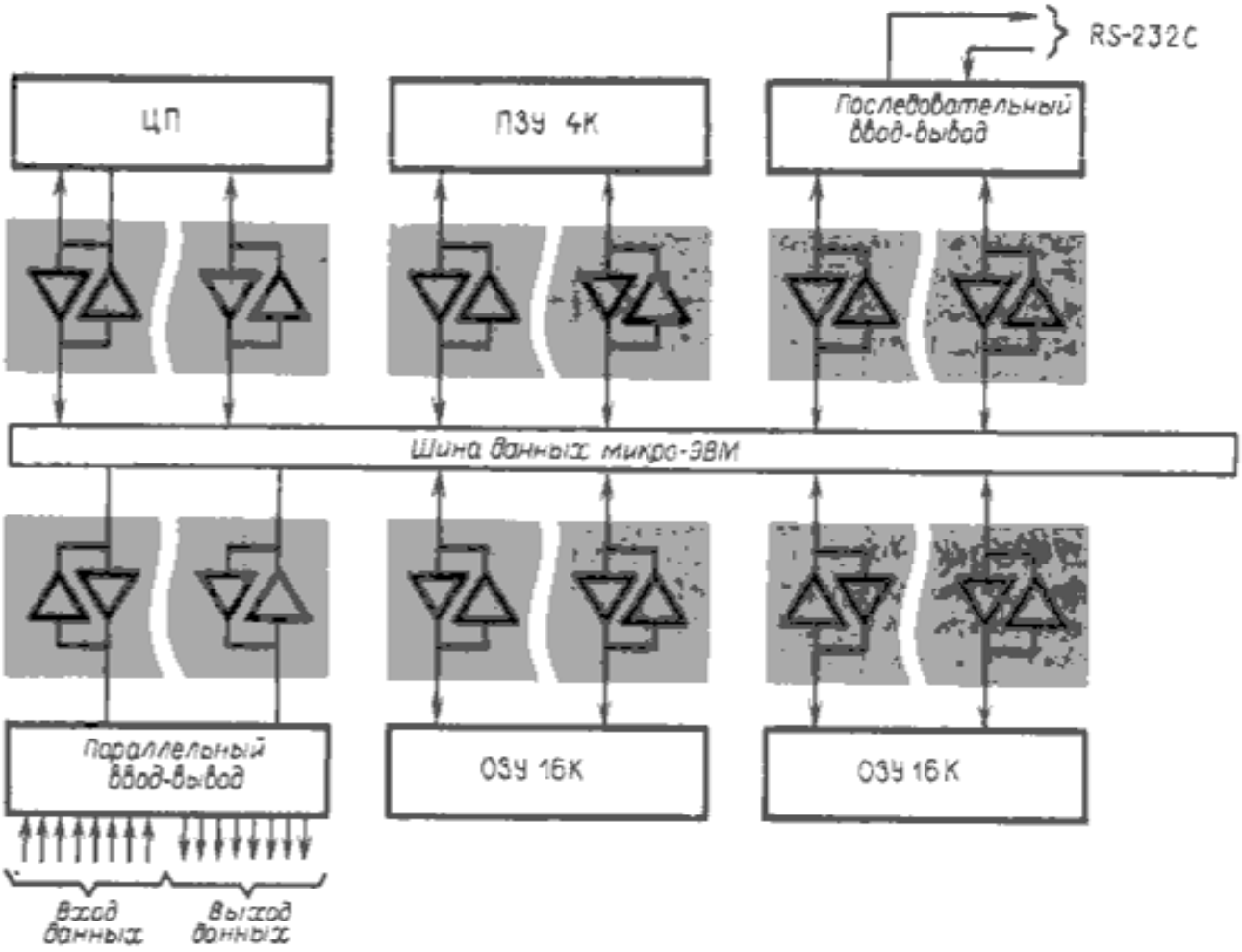


Рис. 14.12. Схема подключения модулей микропроцессорной системы с помощью шинных формирователей. (Каждый модуль, кроме того, имеет буферы на линиях управления и адресных линиях памяти.)

процессорная система, в которой шинные формирователи использованы для подключения к шинам системы различных модулей.

Существует ряд специальных интегральных схем, предназначенных для реализации сложных функций управления. Одной из таких схем является контроллер, выполняющий функции управления в режиме прямого доступа к памяти. По запросу этого контроллера микропроцессор должен перевести в состояние высокого сопротивления линии данных и адресные линии. Затем контроллер выполняет необходимые функции по передаче данных между памятью системы и внешними устройствами. По завершении обмена в режиме прямого доступа к памяти контроллер передает функции управления памятью микропроцессору.

Еще одной специальной интегральной схемой является *шифратор приоритетов*, используемый в системе прерываний микропроцессора. При поступлении запросов на прерывание от различных устройств шифратор приоритетов разрешает прохождение запроса от устройства с более высоким приоритетом. Кроме того, эта схема выдает адресную информацию, с помощью которой микропроцессор выбирает программу обслуживания соответствующего прерывания.

Почти все шифраторы приоритетов прерываний являются программируемыми. Посредством загрузки управляющего слова в регистр состояния модуля шифратора приоритетов осуществляется управление характеристиками и функциями модуля. Таким образом, характеристики и функции рассматриваемого устройства могут оперативно изменяться программным путем во время функционирования системы.

Задания для самопроверки

9. Какое значение в шестнадцатеричном представлении следует помещать в таймер (см. рис. 14.8), на вход которого поступают тактовые импульсы с периодом 1 мс, чтобы он вырабатывал по одному импульсу каждые 1,5 с?

10. Пусть микропроцессор используется для реализации некоторой прикладной задачи. Длина программы составляет 128 байт, а для размещения данных достаточно 64 байт памяти. Микропроцессор имеет один двунаправленный порт ввода-вывода. Может ли быть построена микропроцессорная система, удовлетворяющая указанным требованиям, на основе микропроцессора и модуля, представленного на рис. 14.8? Если это возможно, то укажите функции, выполняемые модулями системы. Если же построение желаемой системы с использованием двух модулей осуществить нельзя, обоснуйте ваши выводы и укажите, какие дополнительные функции и соответствующие аппаратные средства потребуется использовать для обеспечения указанных требований.

11. Микропроцессорная система содержит четыре модуля, аналогичные модулю, представленному на рис. 14.8. Каждый модуль включает ОЗУ, порты ввода-вывода и таймер. В этой системе, кроме того, использовано четыре ПЗУ, схемы которых подобны схеме, представленной на рис. 14.7. Покажите, каким образом эти модули могут быть подключены к шине данных и адресной шине микропроцессора. Если при этом требуются дополнительные логические схемы, то поясните вывод, к которому вы пришли, и изобразите используемые логические элементы на общей схеме.

12. Микропроцессорная система, описанная в предыдущем задании, работает с генератором тактовых импульсов, следующих с частотой 5 МГц. Выход генератора тактовых импульсов соединен с входом «Тактовые импульсы» таймера одного из модулей. Выход этого таймера соединен с входом таймера второго модуля. В таймер первого модуля записывается наибольшее возможное значение. Чему будет равно время между появлениями сигналов прерывания на выходе второго таймера при записи в него наименьшего (наибольшего) возможного значения?

13. Вы работаете с микро-ЭВМ, микропроцессор которой имеет раздельные адресные выводы и выводы шины данных. Шина системы буферирована. а) Сколько шинных формирователей, подобных показанному на рис. 14.11, должно быть установлено на плате ЦП? б) Почему требуется именно такое количество формирователей?

14. Почему модуль, представленный на рис. 14.7, не имеет входа для подачи сигнала «Запись в память»?

14.3. Микро-ЭВМ на одном кристалле

Микро-ЭВМ на одном кристалле имеют сравнительно низкую стоимость, что является следствием высокого развития технологии производства средств вычислительной техники на основе БИС. Следующие элементы непременно входят в состав однокристальной микро-ЭВМ: ЦП, ПЗУ, ОЗУ и средства ввода-вывода. Часто микро-ЭВМ, реализованную на одном кристалле, называют однокристальным микропроцессором. В действительности микро-ЭВМ на одном кристалле представляет собой небольшую вычислительную систему.

В большинстве случаев такие однокристальные вычислительные системы построены с использованием микропроцессора достаточной мощности и ряда вспомогательных блоков, принадлежащих тому же семейству микропроцессоров. Однако это правило не является обязательным.

Часто разработка однокристальных микро-ЭВМ, близких некоторому семейству микропроцессоров, облегчается. Это связано с тем, что для создания однокристальной микро-ЭВМ могут применяться средства проектирования и обслуживания, ориентированные на определенное семейство микропроцессоров. Кроме того, знакомство с микропроцессором, родственным однокристальной микро-ЭВМ, облегчает отладку и эксплуатацию.

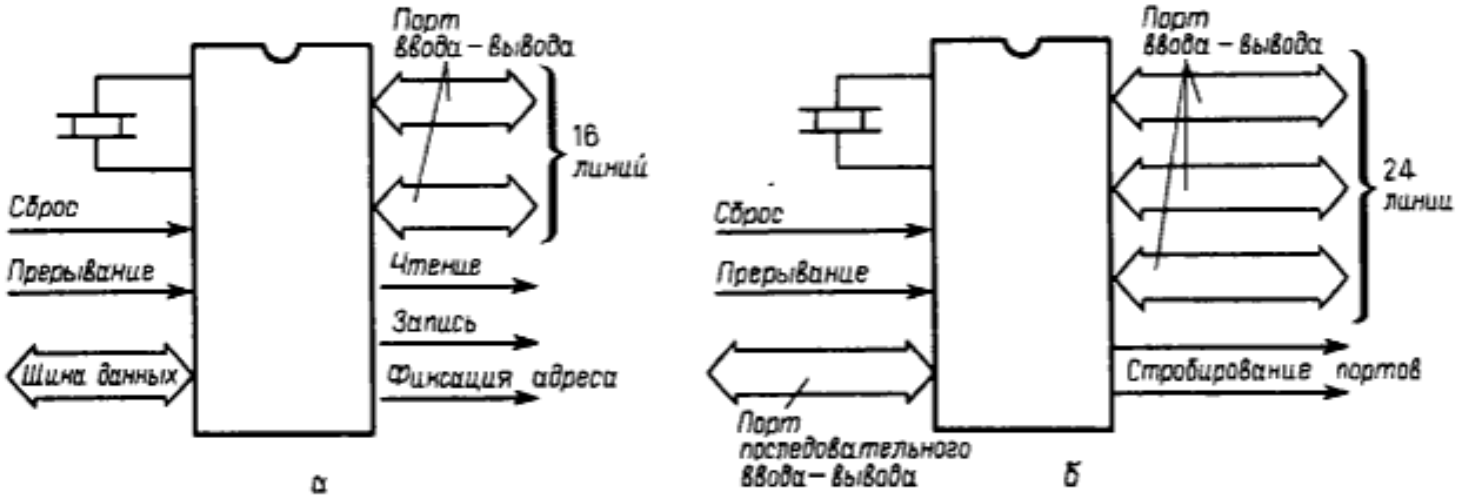


Рис. 14.13. Конфигурация расположения выводов двух однокристальных микро-ЭВМ.

Объем ПЗУ однокристальной микро-ЭВМ обычно составляет 1024 или 2048 байт. Дешевые однокристальные микро-ЭВМ имеют ПЗУ с масочным программированием. Существуют, однако, однокристальные микро-ЭВМ с ПЗУ, в котором предусмотрена возможность стирания информации. Микро-ЭВМ с ПЗУ такого типа в некоторых случаях может устанавливаться непосредственно на панели, предназначенной для аналогичной микро-ЭВМ, имеющей ПЗУ с масочным программированием, т.е. для указанных вариантов реализации микро-ЭВМ выводы имеют одинаковое расположение. Такая совместимость облегчает разработку и развитие технических средств системы. Следует иметь в виду, что микро-ЭВМ с ПЗУ с возможностью стирания информации стоит обычно в 25–50 раз больше, чем аналогичная микро-ЭВМ, включающая ПЗУ с масочным программированием.

Однокристальная микро-ЭВМ имеет в своем составе также ОЗУ, объем которого, как правило, значительно меньше объема ПЗУ. Обычно используются микро-ЭВМ, объем ОЗУ которых равен 64, 128 или 256 байт. Некоторые однокристальные микро-ЭВМ имеют специальные выводы, предназначенные для подключения дополнительных блоков памяти. Однако существуют микро-ЭВМ, объем памяти которых не может быть увеличен. Вообще, выход за пределы объема памяти, предусмотренного проектом, нежелателен. Однокристальные микро-ЭВМ часто используются для решения одной специальной задачи, программная реализация которой требует определенного объема памяти. В некоторых случаях основной памяти оказывается недостаточно, и тогда приходится использовать дополнительную память.

В ОЗУ многих однокристальных микро-ЭВМ не допускается хранение программ. В таких ЭВМ оперативное запоминающее устройство используется только для хранения данных. Все программы записываются в ПЗУ, поскольку выборка команды по адресу, задаваемому содержимым счетчика команд, может осуществляться в большинстве ЭВМ рассматриваемого класса только из ПЗУ. Программы для таких микро-ЭВМ записываются в ПЗУ в процессе его программирования. Практически невозможно внести изменение в программу, хранимую в ПЗУ, однако такое отсутствие гибкости не является большим недостатком, поскольку, как было отмечено выше, однокристальная микро-ЭВМ обычно предназначена для решения одной специальной задачи.

Большинство микро-ЭВМ обладает значительными возможностями по вводу-выводу данных. На рис. 14.13 представлено расположение выводов двух незначительно отличающихся микро-ЭВМ. Однокристальный микропроцессор, изображенный на рис. 14.13, *а*, имеет 16 двунаправленных линий ввода-вывода и 8-разрядную шину данных. Эта микро-ЭВМ допускает расширение памяти. На рис. 14.13, *б* изображено расположение выводов однокристальной микро-ЭВМ, которая содержит три 8-разрядных порта параллельного ввода-вывода и порт последовательного ввода-вывода, снабженный необходимыми схемами упра-

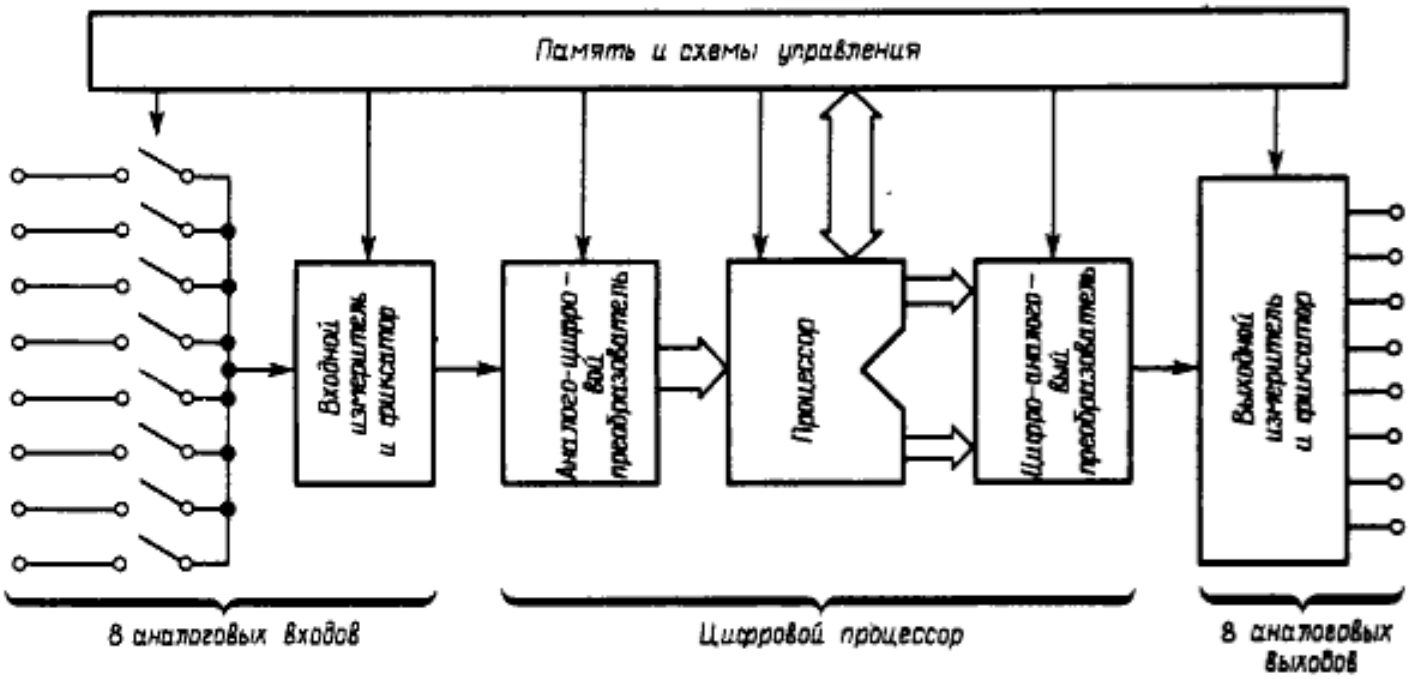


Рис. 14.14. Схема аналого-цифровой микро-ЭВМ.

вления модемом. Обе микро-ЭВМ имеют внутренние таймеры.

На рис. 14.14 представлена схема однокристальной микро-ЭВМ другого типа. Это *аналого-цифровая* микро-ЭВМ, предназначенная для выполнения цифровой обработки аналоговых сигналов. Данная микро-ЭВМ имеет аналоговый вход и аналоговый выход. На входе аналого-цифровой преобразователь производит преобразование аналоговых входных сигналов в слова данных, являющиеся цифровым представлением сигналов. На выходе с помощью цифро-аналогового преобразователя осуществляется преобразование сигналов из цифровой формы в аналоговую.

Как входной, так и выходной порты имеют по восемь линий. Восьмиканальный аналоговый мультиплексор и схема измерения и фиксации значений аналогового сигнала могут использоваться для преобразования сигналов, действующих на восьми аналоговых входах, в сигналы, передаваемые по одному аналоговому каналу.

Для обработки сигналов, представленных в цифровой форме, используется специальный ЦП. Он выполняет команды, хранимые в памяти аналого-цифровой микро-ЭВМ.

Так как аналого-цифровой преобразователь проводит замеры с очень большой частотой, он обеспечивает достаточно точное цифровое представление сигнала. Такой способ взятия замеров непрерывного сигнала приемлем даже тогда, когда на каждом аналоговом входе действует сравнительно высокочастотный сигнал. Согласно теории дискретного представления непрерывных величин, для обеспечения точного представления входного непрерывного сигнала в течение каждого периода, соответствующего наибольшей частоте в спектре входного сигнала, требуется выполнить по крайней мере два замера. На практике в течение каждого периода колебаний, соответствующего наибольшей частоте в спектре сигнала, производится пять или более измерений. На входе и выходе аналого-цифровой микро-ЭВМ используются фильтры, которые обеспечивают правильное представление исходного и выходного сигналов.

Аналого-цифровые микро-ЭВМ находят широкое применение. Это обусловлено тем, что аналоговый фильтр, имеющий строгое математическое описание, не позволяет практически получить реактивную и активную составляющие сигнала, точно соответствующие теории. Известно, что классические аналоговые фильтры реализуются с использованием резисторов, индуктивностей и конденсаторов. С помощью такого фильтра часто бывает трудно или даже невозможно получить действительные составляющие, точно совпадающие с теоретическими значениями. В связи с этим при проектировании фильтров часто принимают компромиссное решение. Используя

аналого-цифровую микро-ЭВМ, можно производить фильтрацию, качество которой будет хорошо согласовано с теорией фильтрации. Таким образом, после преобразования аналогового сигнала в цифровую форму с помощью микро-ЭВМ реализуются все преимущества цифровой фильтрации. Цифровые схемы микро-ЭВМ обеспечивают обработку цифровых данных, соответствующих аналоговому сигналу, по алгоритмам преобразования реактивной и активной составляющих сигнала, определенным в теории фильтрации. После осуществления цифровой обработки сигнал претерпевает преобразование из цифровой формы в аналоговую.

Аналого-цифровая микро-ЭВМ является важным элементом систем распознавания и синтеза речи. При решении задачи распознавания речевых сигналов требуется производить деление сигнала на его составляющие. Каждая составляющая должна быть проанализирована; такой анализ целесообразно выполнять с помощью аналого-цифровой микро-ЭВМ. При синтезе речевой сигнал должен быть сформирован с учетом частотных, амплитудных и временны́х характеристик. Это обеспечит получение голоса, близкого к человеческому. Синтез речи представляет собой процесс, легко выполняемый с помощью микро-ЭВМ.

Задания для самопроверки

15. Некоторая однокристальная микро-ЭВМ выпускается в двух вариантах – с ПЗУ и ПЗУ с возможностью стирания информации. Почему целесообразно производить оба типа микро-ЭВМ?

16. Автономная система контроля, основанная на микропроцессоре, встроена в большую насосную станцию. Известно, что для осуществления функций контроля система должна производить анализ 32 цифровых и 8 аналоговых сигналов. Вследствие развития системы контроля потребовалось осуществлять анализ еще 8 цифровых и 6 аналоговых сигналов. Можно ли создать такую систему на основе однокристальной микро-ЭВМ или для этого потребуется использовать микро-ЭВМ на нескольких кристаллах? Объясните ваше решение.

17. Преобразователь аналогового сигнала в цифровую форму производит пять замеров за период, соответствующий максимальной частоте в спектре аналогового сигнала. Какое количество измерений в секунду делает преобразователь, входящий в аналого-цифровую микро-ЭВМ, если максимальная частота сигнала на любом входе равна 5 кГц?

18. Какая из перечисленных ниже особенностей однокристального микропроцессора является наиболее важной:

а) структура прерываний;

б) большой объем памяти;

в) низкая стоимость;

г) наличие средств установки в начальное состояние?

19. Производится сопоставление однокристальной микро-ЭВМ и микропроцессора. Какая из следующих особенностей присуща только микро-ЭВМ:

а) наличие генератора тактовых импульсов с кварцевой стабилизацией частоты;

б) наличие большого числа портов ввода-вывода;

в) наличие таймера;

г) возможность расширения памяти?

20. Какую из следующих функций выполняет схема измерения и фиксации, входящая в состав аналого-цифровой микро-ЭВМ:

а) фиксация замера значения входного сигнала до тех пор, пока аналого-цифровой преобразователь не выполнит преобразование сигнала в цифровую форму;

б) фиксация для каждой линии выходного сигнала цифро-аналогового преобразователя, в то время как преобразователь вырабатывает другие сигналы;

в) выбор желаемой точки (обычно пиковой) входного сигнала, используемой для измерения значения и его преобразования в цифровую форму;

г) выдача 8-разрядного числа в аналого-цифровой преобразователь?

21. В большинстве однокристальных микро-ЭВМ двунаправленные порты ввода-вывода программируются либо самим пользователем, либо при изготовлении ПЗУ с масочным программированием. Какие из приведенных ниже объяснений правильно истолковывают целесообразность использования программируемых портов ввода-вывода?

а) При использовании программируемых портов пользователь может сам решать, является ли некоторый порт входным или выходным;

б) уменьшаются затраты на программирование по сравнению с подобными затратами, которые имели бы место при использовании ПЗУ с возможностью стирания информации;

в) обеспечивается защита входных линий во время выполнения операции прямого доступа к памяти;

г) справедливы все перечисленные выше объяснения.

14.4. Средства контроля и отладки микропроцессорных систем

На каждом новом этапе развития электроники возникают свои задачи по созданию измерительной и контрольной аппаратуры. Например, создание логических схем с эмиттерными связями привело к необходимости разработки специального высокочастотного осциллографа. Особенно много новых технических средств, используемых на стадии проектирования и эксплуатации, появилось в связи с созданием микропроцессоров.

В настоящем разделе будут описаны некоторые контрольно-измерительные приборы, которые могут потребоваться при работе с микропроцессорными системами. Очевидно, что кроме специальных приборов для выполнения контрольно-измерительных операций используются и такие широко применяемые инструменты, как осциллограф, счетчики, вольтметры и генераторы импульсов.

Одной из основных проблем, возникших в связи с созданием микропроцессоров и микропроцессорных систем, является проблема разработки и тестирования таких систем. Представьте себе, что по готовым принципиальным схемам при наличии программного обеспечения, представленного в виде распечаток программ, необходимо получить действующую микропроцессорную систему. Очевидно, можно сделать макет аппаратных средств и поместить программы в ПЗУ, а затем проверить работоспособность системы. Весьма вероятно, что разработанная таким образом микропроцессорная система работать не будет. Тогда придется заняться отысканием неисправности.

Существует специальная система, которую называют *системой разработки микропроцессоров* (*СРМ*). Ее можно подключать к создаваемой системе посредством того же разъема, который предназначен для разрабатываемого микропроцессора. Упомянутая система имеет на другом конце этого соединения микропроцессор. Последний подключается к дополнительной памяти и специальным логическим схемам с целью обеспечения регистрации данных на шине данных испытываемого микропроцессора во время его тестирования. СРМ обычно управляется посредством второго микропроцессора.

Система разработки микропроцессоров может поместить вашу программу в память разрабатываемой системы либо в собственную память СРМ. При работе система выполняет определенные команды и выдает сообщения на видеотерминал. Кроме того, систему разработки микропроцессоров можно использовать как средство разработки программного обеспечения. Она предоставляет разработчикам эффективную программу-редактор, облегчающую написание исходных текстов программ на языке ассемблера. Эта система располагает также ассемблером, с помощью которого выполняется ассемблирование исходного текста программы и получение программы в машинном коде вашего микропроцессора.

Созданная с помощью ассемблера СРМ объектная программа может быть загружена в память разрабатываемой системы. Теперь можно производить пошаговое выполнение программы. СРМ также позволяет задавать точки останова программы. Благодаря такой возможности несколько команд выполняется при номинальной скорости работы микропроцессора, а затем в указанной точке происходит останов микропроцессора. При каждом останове можно убедиться в правильности действий, выполненных микропроцессором. Естественно, система разработки позволяет осуществлять и «прогон» полной программы при номинальной скорости работы микропроцессора.

В некоторых СРМ реализованы языки высокого уровня, например ПАСКАЛЬ или БЕЙСИК. При этом для программ, написанных на определенном языке высокого уровня, СРМ вырабатывает соответствующий объектный код для вашего микропроцессора.

Можно сказать, что СРМ позволяет отлаживать аппаратные и программные средства в «режиме реального времени» или по крайней мере делать это по возможности оперативно. Мы говорим здесь о «реальном времени», поскольку микропроцессор выполняет команды при своей рабочей скорости. Замедление выполнения команд обусловливается процессом тестирования, который может осуществляться, например, в режиме пошагового выполнения команд. Тестирование в режиме реального времени помогает обнаруживать неисправности, которые проявляются при работе микропроцессора с максимальной скоростью. Если же микропроцессор работает при пониженной скорости, то такие неисправности часто остаются незамеченными.

Предположим, что ваша микропроцессорная система вводит данные через порт параллельного ввода-вывода. Каждый раз, когда система выполняет ввод в пошаговом режиме, она действует правильно. Однако при попытке ввода данных при рабочей скорости происходит зацикливание системы. Система разработки поможет решить задачу отыскания возникшей неисправности. С помощью СРМ можно проверить правильность передачи данных по адресной шине и шине данных микропроцессора во время выполнения нескольких команд, предшествующих команде ввода, и нескольких команд, следующих за командой ввода, при выполнении которой обнаруживается неисправность.

Используя СРМ, можно начать проверку правильности каждой передачи по шине сразу после некоторой пусковой точки. Обычно пусковой точкой является момент начала выполнения определенной команды или момент начала адресации некоторой области памяти. Все данные, накопленные при проверке правильности передач по шине в памяти СРМ, позже могут быть проанализированы. В нашем случае, например, может быть обнаружено, что при работе микропроцессора вследствие ошибок синхронизации регистр состояния порта ввода-вывода устанавливается в состояние 0. При нулевом содержимом регистра состояния данные не могут быть введены в микропроцессор.

Кроме описанной возможности проверки правильности передачи данных СРМ обеспечивает выполнение многих других функций. Например, по завершении выполнения каждой команды СРМ может выдать на экран дисплея содержимое каждого регистра ЦП и содержимое нескольких выбранных областей памяти. Система разработки не может производить указанные действия при рабочей скорости, но тем не менее для некоторых ограниченных участков программы эти операции могут выполняться при достаточно высокой скорости.

Другой особенностью СРМ является возможность подачи сложных комбинаций тестовых сигналов, что облегчает процесс отыскания неисправностей в тестируемой системе. Пожалуй, одним из важнейших достоинств СРМ является то, что она позволяет весьма быстро выполнять исправления ошибок в программе. После обнаружения ошибки в программе для ее исправления используется редактор. Таким образом, программа может быть быстро скорректирована, соответствующим образом документирована и подготовлена к дальнейшему тестированию.

Для наладки микропроцессорных систем широко используется *логический анализатор* – прибор, который значительно лучше подходит для этой цели, чем обычный осциллограф. В логическом анализаторе имеется визуальное устройство отображения, подобное применяемому в осциллографах. Однако в принципе действия логического анализатора и осциллографа есть значительное различие. Во-первых, логический анализатор обеспечивает отображение сигналов, действующих на большом числе входов. Существующие логические анализаторы, применяемые для проектирования и эксплуатации микропроцессоров, могут одновременно отображать 16–32 сигнала. Во-вторых, входы логического анализатора реагируют на логические уровни сигналов, а не на аналоговые сигналы. Это позво-

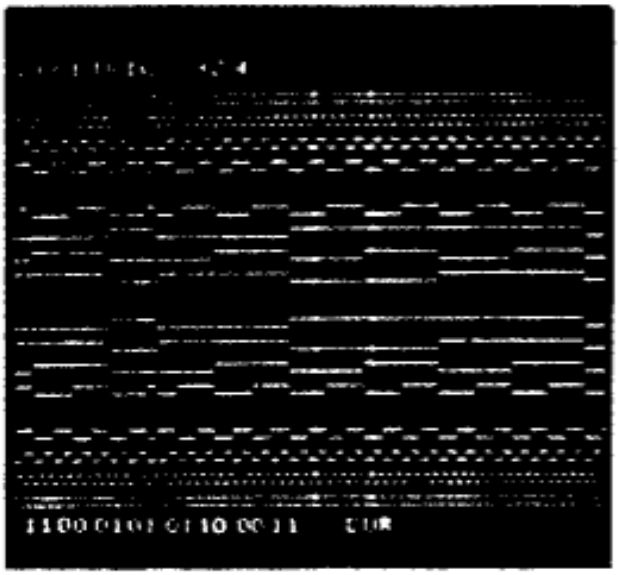


Рис. 14.15. Пример временно́го отображения информации на дисплее логического анализатора. (С разрешения фирмы Tektronix.)

Рис. 14.16. Пример отображения логического состояния на дисплее логического анализатора (С разрешения фирмы Tektronix.)

ляет логическому анализатору отображать состояние своих входов в виде логических нулей и единиц.

Обычно логические анализаторы обладают весьма универсальными и гибкими возможностями по вводу данных и запуску. Так, ввод данных можно осуществлять 16- или 32-разрядными словами, которые могут подаваться параллельно на 16 или 32 входа. Кроме того, при необходимости можно производить ввод данных 16- или 32-разрядными словами, подаваемыми последовательно на один из 16 или 32 входов. С целью отображения связанных событий логический анализатор может быть запущен и затем переведен в состояние ожидания до тех пор, пока не будет передано заранее установленное количество импульсов или слов. Данные со входов логического анализатора записываются в его память. Например, в память 32-разрядного логического анализатора может быть помещено 256 32-разрядных слов.

Поскольку входные данные регистрируются в памяти анализатора, для отображения на экране могут быть использованы как данные, сформированные перед запуском, так и данные, сформированные после запуска анализатора. Например, если анализатор был настроен на прием одного слова, то 255 слов, зарегистрированных ранее, все еще находятся в памяти. Анализатору можно указать хранить эти данные и не воспринимать данные, поступающие после его возможного запуска. Подобным же образом можно настроить анализатор на прием 256 слов, поступивших после его запуска. Обычный осциллограф таких возможностей не обеспечивает.

Логический анализатор имеет два режима отображения информации – *временно́е отображение* и *отображение состояния*. Соответствующие примеры представлены на рис. 14.15 и 14.16.

Пример временно́го отображения информации на экране, представленный на рис. 14.15, напоминает изображение на экране обычного многолучевого осциллографа. Данные для временно́го отображения поступают не со входов анализатора, а из его памяти, где они хранятся. Анализатор производит измерение сигналов на своих входах через равные временны́е интервалы при синхронизации от системного генератора тактовых импульсов. Обычно частота, с которой анализатор производит измерения, в пять раз выше, чем наивысшая частота генератора тактовых импульсов, используемого в системе. Частота измерений, выполняемых анализатором в режиме временно́го отображения, определяет его временну́ю разрешающую способность. Например, если период выборки значений сигналов составляет 100 нс, то с помощью анализатора можно измерять сигналы и регистрировать события, длительность которых превышает 100 нс.

Анализатор используется для измерения

временны́х характеристик сигналов в некоторой системе. Посмотрев на экран, вы без труда определите, какое событие в системе произошло первым. Кроме того, анализатор позволяет измерять интервалы времени между моментами наступления различных событий.

В режиме отображения состояния анализатор обеспечивает визуальное наблюдение за логическими значениями сигналов, действующих на его входах. Отображаемая при этом информация обновляется каждый раз, когда изменяется состояние на выбранных входах анализатора. Таким образом, изменение состояния в измеряемой системе обусловливает изменение входных данных анализатора. Данные на экране анализатора обычно представляются в двоичной, восьме- или шестнадцатеричной форме. На рис. 14.16 показано изображение на экране в двоичном коде логического состояния сигналов.

Режим отображения логического состояния очень полезен при наблюдении за передачами данных по шине микропроцессора. Каждое слово данных, передаваемое по шине, может быть представлено на экране. Анализатор реагирует только на изменение логических значений сигналов на его входах; он индифферентен к длительности временны́х интервалов между моментами изменений состояний. В данном режиме анализатор лишь обеспечивает регистрацию последовательности изменяющихся сигналов.

При проектировании и эксплуатации микропроцессорных систем используются еще два сравнительно простых, но очень полезных контрольно-измерительных прибора – *логический пробник* и *сигнатурный анализатор*.

Логический пробник осуществляет детектирование логического уровня сигнала и индикацию результата анализа. С помощью логического пробника определяется, в каком состоянии – логического 0 или 1 – находится проверяемая схема. Некоторые логические пробники имеют схему расширения импульса. Схема расширения импульса позволяет логическому пробнику регистрировать и индицировать появление каждого, даже очень короткого импульса.

Логический пробник является хорошим средством для контроля портов ввода-вывода микропроцессора; линий, по которым передаются признаки состояния, и других точек системы, состояние которых изменяется сравнительно редко. Например, может быть написана такая программа, согласно которой микропроцессор будет многократно читать содержимое одной и той же области памяти. Во время выполнения этой программы с помощью логического пробника можно проверить, подается ли импульс «Чтение памяти» во все определенные точки системы. В некоторых микропроцессорных системах при продолжительном выполнении циклической программы, в которой осуществляется чтение одной и той же области памяти, удается проверить правильность прохождения сигналов по шине данных. Существуют, однако, системы, в которых выполнить с помощью логического пробника такого рода проверку не удается.

Сигнатурный анализатор по существу представляет собой логический пробник, снабженный памятью. Такой памятью является 16-разрядный сдвиговый регистр. Сигнатурный анализатор применяется для контроля данных на одной линии. Для подлежащей тестированию микропроцессорной системы должны быть написаны специальные диагностические программы. Эти программы должны обеспечить передачу по проверяемым линиям некоторого повторяемого последовательного потока данных. Сигнатурный анализатор получает данные с проверяемой линии и отображает «сигнатуру», или код, который в нашем примере является четырехразрядным шестнадцатеричным числом, с помощью специального индикатора.

Правильный код линии заранее определяется и фиксируется. При проверке производят сравнение числа, отображаемого индикатором, с предварительно определенным кодом линии. Если сравниваемые числа не совпадают, то велика вероятность того, что в анализируемой последовательности битов имеется ошибка. Сигнатурный анализатор позволяет обнаруживать одиночные ошибки в потоке данных. Такая возможность очень полезна при контроле систем, работу которых нельзя остановить

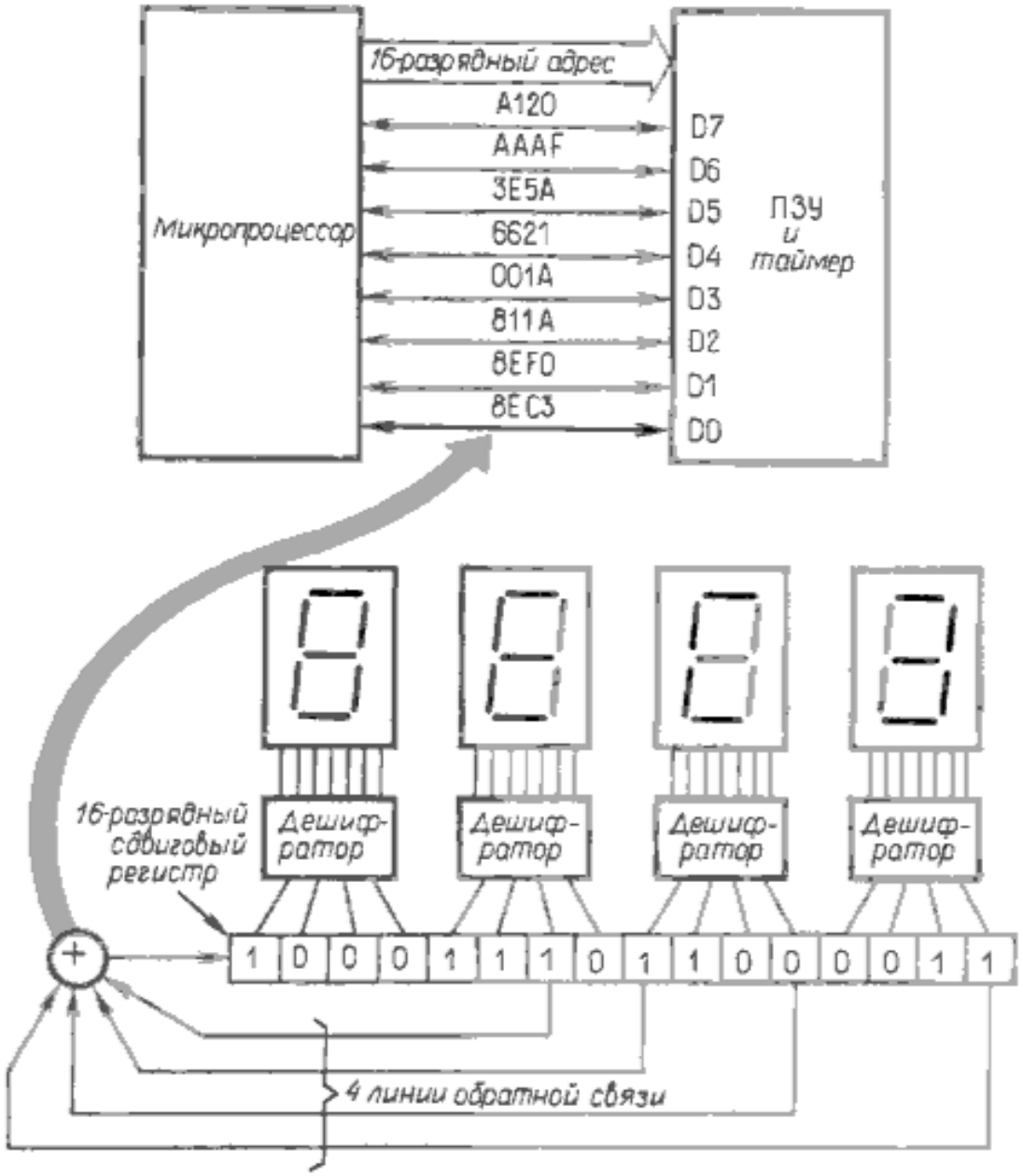


Рис. 14.17. Схема подключения сигнатурного анализатора для анализа линии D0 шины данных микропроцессора. (Над каждой линией шины данных указан соответствующий код.)

или которые невозможно перевести в пошаговый режим.

На рис. 14.17 показано, как сигнатурный анализатор отображает код. Предполагается, что микропроцессор выполняет диагностическую программу. Работая по такой программе, микропроцессор передает по каждой линии шины данных определенную повторяющуюся последовательность логических 0 и 1. Это достигается повторяющейся передачей данных по шине данных в прямом и обратном направлениях. Если диагностическая программа работает правильно, то каждая линия шины данных будет иметь свой постоянный код. Поток данных, передаваемых по линии D0, поступает на вход сигнатурного анализатора. Входной поток поступает на сумматор, на который подаются также по линиям обратной связи сигналы, вырабатываемые сдвиговым регистром. Сумма подается на вход сдвигового регистра. Параллельные выходы сдвигового регистра связаны с дешифраторами. Сигналы с выходов дешифраторов поступают на семисегментные индикаторы, которые и отображают код потока.

Отображаемый на индикаторе код сравнивается с кодом, изображенным на схеме. Очевидно, что каждая линия шины данных имеет свой код. При отыскании неисправностей в этой микропроцессорной системе необходимо последовательно использовать сигнатурный анализатор для проверки каждой линии. Одиночные ошибки в потоке битов будут вызывать искажение сигнатуры, обнаружив которое, вы установите место неисправности.

Задания для самопроверки

22. Какое из приведенных ниже объяснений является правильным объяснением того, почему для отладки цифровых систем предпочтительнее пользоваться логическим анализатором, а не осциллографом:

а) люминофор осциллографа не обеспечивает требуемого качества изображения;

б) с помощью осциллографа нельзя отображать шестнадцатеричные числа;

в) осциллограф не имеет памяти;

г) осциллограф не обладает требуемой частотной характеристикой?

23. Одной из отличительных особенностей СРМ является возможность делать «моментальные снимки» содержимого всех регистров микропроцессора после выполнения каждой команды. При каких из перечисленных ниже условий реализуется указанная возможность:

а) при работе в реальном времени;

б) при работе микропроцессора при пониженной скорости;

в) при работе в пошаговом режиме;

г) при использовании логического анализатора?

24. Какая из перечисленных ниже функций системы разработки микропроцессоров является основной:

а) разработка программного обеспечения микропроцессора;

б) регистрация данных, передаваемых по шине;

в) отображение состояния регистров ЦП;

г) оперативная отладка программно-аппаратного обеспечения микропроцессорных систем?

25. Некоторая микропроцессорная система проектировалась без учета применения при ее эксплуатации сигнатурного анализа. По какой из перечисленных ниже причин затруднительно использовать сигнатурный анализ для контроля и диагностики этой системы:

а) в системе не будут реализованы правильные логические уровни сигналов;

б) потребуется предварительно определить все необходимые коды и разработать специальную диагностическую программу;

в) сигнатурный анализатор может быть применен при работе системы при пониженной скорости;

г) для реализации процедур сигнатурного анализа потребуется специальное оборудование?

26. Какой из перечисленных ниже режимов работы логического анализатора близок к режиму работы стандартного многолучевого осциллографа:

а) режим отображения состояния;

б) режим временно́го отображения;

в) режим предварительного запуска;

г) режим шестнадцатеричного представления отображения информации?

27. С помощью каких элементов [а) ЭЛТ, б) цифровой памяти, в) средств селективного запуска, г) большого количества входов] в логическом анализаторе обеспечивается хранение проведенных измерений входных сигналов?

Упражнения

14.1. Сколько выводов имеют корпусы, обычно используемые для микропроцессоров?

14.2. Какие из приведенных ниже обоснований оправдывают мультиплексирование линий передачи адреса и данных:

а) сокращается количество выводов для подачи синхронизирующих сигналов;

б) уменьшается сложность системы адресации периферийных устройств;

в) сокращается количество выводов микропроцессора?

14.3. Шестнадцатиразрядный микропроцессор имеет 40 выводов. Для передачи адреса и данных используется мультиплексированная шина. Этот микропроцессор может непосредственно адресовать 1 048 576 областей памяти. Первая линия передачи адреса и данных имеет обозначение AD0. Укажите обозначения для остальных линий этой шины.

14.4. Какие дополнительные сигналы необходимы для организации передачи информации по мультиплексированной шине, используемой для передачи адреса и данных?

14.5. Сформулируйте требования, которым должен удовлетворять микропроцессор, совместимый с ТТЛ-схемами?

14.6. Каким образом используются сигналы о состоянии ЦП?

14.7. Как используются сигналы синхронизации микропроцессора?

14.8. Тактовые импульсы часто формируются с помощью внутренних схем микропроцессора. Какими особенностями обладают тактовые им-

пульсы, используемые для синхронизации микропроцессоров?

14.9. Какие средства обычно используются для задания опорной частоты генераторов тактовых импульсов?

14.10. Назовите известные вам способы генерирования сигналов заданной частоты, используемых для формирования последовательностей тактовых импульсов микропроцессоров.

14.11. Какие требования предъявляются к источникам питания микропроцессоров? Какова величина обычно используемого напряжения питания? Укажите, какая потребляемая мощность характерна для современных микропроцессоров.

14.12. С какой целью подается и каким образом реализуется в микропроцессорах сигнал «Сброс»?

14.13. Почему для сигнала начальной установки используется обозначение «$\overline{\text{Сброс}}$», а не «Сброс»?

14.14. Какой из перечисленных ниже входов не является входом для запроса на прерывание микропроцессора:

а) «$\overline{\text{Сброс}}$»;

б) «Немаскируемое прерывание»;

в) «$\overline{\text{Запрос на прямой доступ к памяти}}$»;

г) «Маскируемое прерывание»?

14.15. При выполнении каких из перечисленных ниже условий микропроцессор вырабатывает сигнал подтверждения прерывания:

а) микропроцессор готов к выполнению прямого доступа к памяти;

б) микропроцессор готов к обработке прерывания;

в) микропроцессор выполняет текущую команду;

г) микропроцессор готов к выполнению следующей команды?

14.16. Какие из перечисленных ниже действий выполняются после поступления запроса на организацию режима прямого доступа к памяти:

а) адресная шина переводится в состояние высокого сопротивления;

б) подается сигнал подтверждения принятия запроса на прямой доступ к памяти;

в) шина данных переводится в состояние высокого сопротивления?

14.17. Во время реализации режима прямого доступа к памяти микропроцессор не функционирует. Укажите, после каких из перечисленных ниже действий возобновляется работа микропроцессора:

а) обращение к памяти;

б) освобождение шины данных;

в) снятие сигнала «$\overline{\text{Запрос на прямой доступ к памяти}}$».

14.18. Путем подачи сигнала на вход «$\overline{\text{Готово}}$» микропроцессора 8085 медленнодействующие внешние устройства могут сделать запрос на перевод микропроцессора в одно из следующих состояний. Укажите это состояние:

а) состояние ожидания в течение нескольких циклов ЦП;

б) ожидание завершения выполнения операции прямого доступа к памяти;

в) прерывание выполнения текущей команды;

г) формирование сигнала чтения из памяти.

14.19. Устройства параллельного ввода-вывода обычно разрабатываются с применением ТТЛ-схем средней и высокой степени интеграции. Какие из перечисленных ниже средств могут также использоваться для этой цели:

а) комплементарные МОП-схемы;

б) схемы параллельного ввода данных;

в) универсальные порты ввода-вывода, относящиеся к особому семейству микропроцессоров;

г) схемы параллельного вывода данных?

14.20. Какие из перечисленных ниже элементов могут входить в некоторый блок, принадлежащий определенному семейству микропроцессоров:

а) логические схемы и дешифратор адреса, входящие в шинный интерфейс микропроцессора;

б) порт последовательного ввода-вывода со схемой проверки на четность;

в) ОЗУ емкостью не менее 256К;

г) ПЗУ с возможностью стирания информации?

14.21. Чем объясняется преимущество аппаратного таймера по сравнению с программным способом задания временных интервалов?

14.22. Опишите принцип действия аппаратного таймера. В каких режимах он может функционировать?

14.23. На основе многих 8-разрядных микропроцессоров в рамках определенного микропроцессорного семейства разрабатываются микропроцессорные системы на «трех кристаллах», обладающие минимальными возможностями. Какие основные функции должны быть предусмотрены в такой системе и как эти функции распределены между тремя кристаллами?

14.24. Для большинства датчиков времени в качестве тактовых сигналов служат сигналы от источника частотой 60 Гц. Почему использование сигналов такой частоты является рациональным? Что произойдет, если для указанной цели будут использованы сигналы частотой 50 Гц?

14.25. Объясните принцип действия и назначение шинных формирователей.

14.26. Изготовители микропроцессоров часто указывают, что модули, предназначенные для

использования с микропроцессорами определенного семейства, полностью программируемы. Что при этом имеется в виду?

14.27. Какое влияние должно оказать использование полностью программируемых периферийных устройств микропроцессорной системы на программы, используемые при запуске микропроцессорной системы?

14.28. Почему термин «однокристальный микропроцессор» является некорректным?

14.29. Какой объем ОЗУ имеют типичные однокристальные микро-ЭВМ?

14.30. Какие из указанных ниже элементов обычно входят в однокристальную микро-ЭВМ:

а) порты последовательного ввода-вывода;

б) много портов параллельного ввода-вывода;

в) ЦП;

г) ОЗУ?

14.31. На входе аналого-цифровой микро-ЭВМ функционирует аналоговый мультиплексор. С каким из перечисленных ниже устройств соединяется мультиплексор:

а) цифро-аналоговый преобразователь;

б) аналого-цифровой преобразователь;

в) таймер?

14.32. Какие из перечисленных ниже особенностей применения присущи командам аналого-цифровой микро-ЭВМ:

а) использование 8-разрядных слов данных;

б) использование памяти большого объема;

в) использование для обработки сигналов?

14.33. Каким образом определяется минимально допустимое количество замеров, которое необходимо произвести при преобразовании аналогового сигнала в цифровую форму?

14.34. Какими преимуществами обладает аналого-цифровая микро-ЭВМ, используемая для фильтрации сигналов, по сравнению с применением для этой цели аналогового RC-фильтра?

14.35. Для обеспечения функционирования каких из перечисленных ниже средств необходим микропроцессор:

а) система разработки микропроцессоров;

б) логический анализатор;

в) сигнатурный анализатор?

14.36. Перечислите функции, которые могут выполнять СРМ.

14.37. Каково основное назначение СРМ?

14.38. Каким образом с помощью СРМ проверяется правильность выполнения операций микропроцессора?

14.39. Какие режимы отображения информации используются в логическом анализаторе? В чем состоит их различие?

14.40. Какие предусмотрены возможности по отображению информации, хранимой в памяти логического анализатора?

14.41. Каковы условия использования сигнатурного анализатора?

14.42. В чем состоит основное отличие сигнатурного анализатора от логического пробника?

Ответы на вопросы заданий для самопроверки

1. Благодаря этому можно обеспечить желаемую реакцию микропроцессора при одновременном поступлении запросов на прерывание.
2. При использовании «отрицательной логики» легко обеспечивается подача сигнала на некоторую линию несколькими схемами с открытым коллектором. На рис. 14.18 представлена схема формирования сигналов по данному способу. Название «отрицательная логика» появилось в связи с тем, что состоянию логической 1, или активному состоянию линии некоторой шины, в данном случае соответствует меньший из двух возможных уровней сигнала.
3. Любая программа, в которой для определения длительности некоторого интервала времени используются операции увеличения или уменьшения содержимого некоторого регистра.
4. г. 5. б. 6. г. 7. б. 8. г.
9. В таймер потребуется поместить величину 05DC.
10. Используя имеющиеся средства, построить микропроцессорную систему не удастся, так как в ней будет отсутствовать ПЗУ, необходимое для хранения программ. ОЗУ для этой цели не подходит. В связи с этим необходимо добавить по крайней мере один модуль ПЗУ и, по-видимому, некоторый дешифратор адреса.
11. См. рис. 14.19. Адресные линии A0–A10 ПЗУ и адресные линии A0–A7 ОЗУ соединены

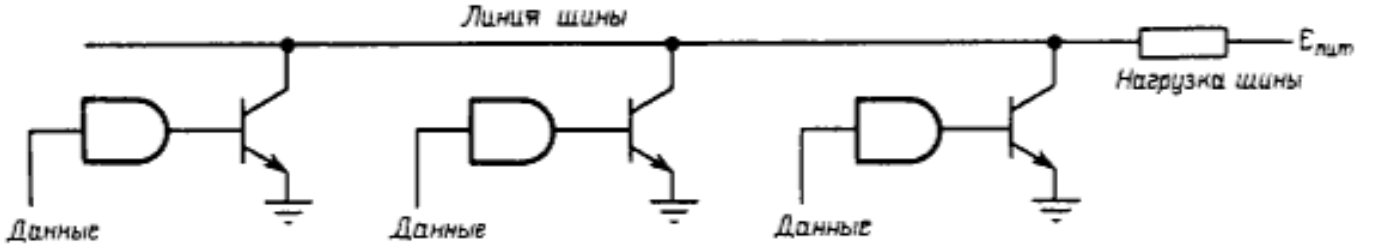


Рис. 14.18. Схема активирования линии с использованием «отрицательной логики».

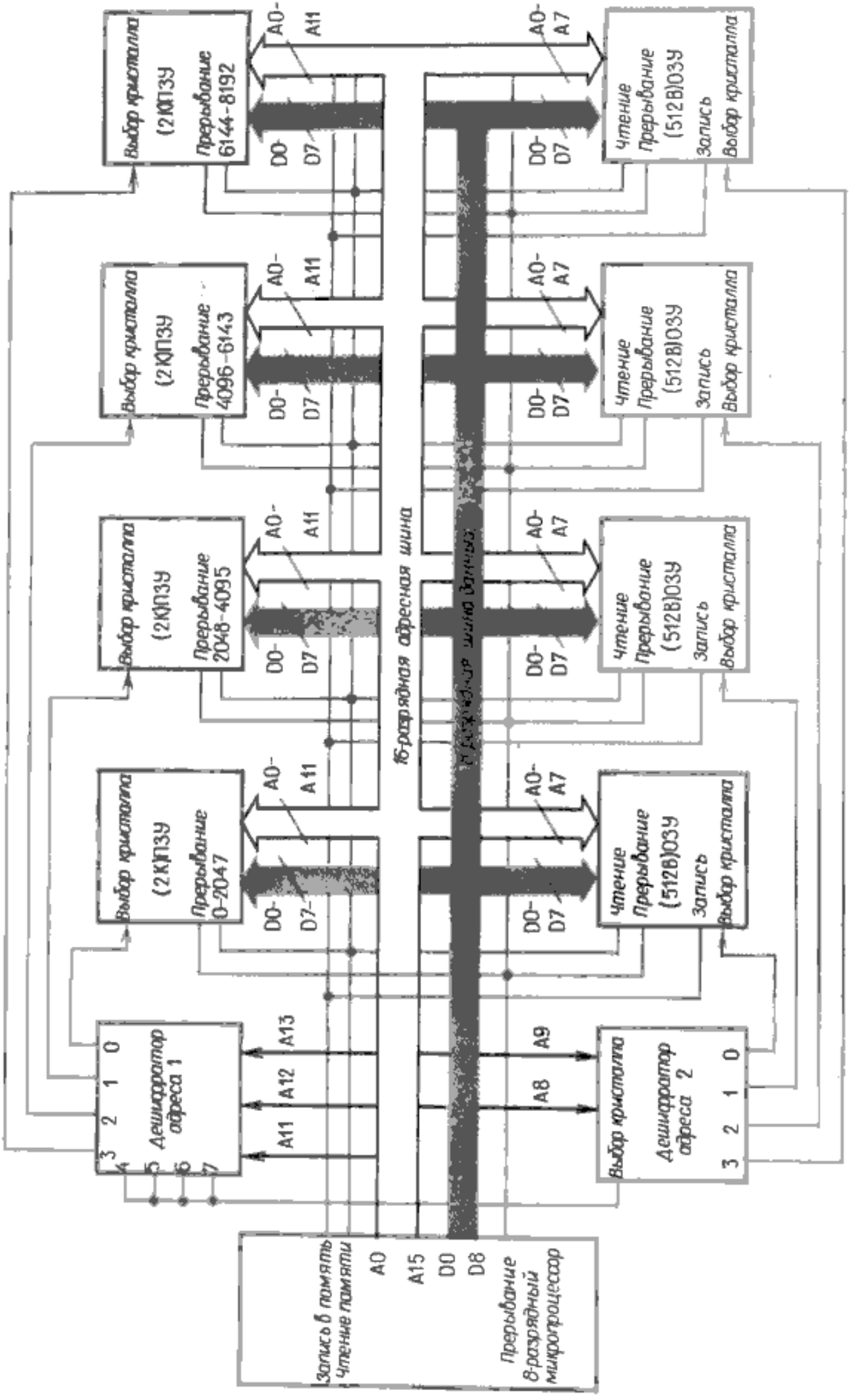


Рис. 14.19. Схема микропроцессорной системы. (К ответу на вопрос 11.)

с 16-разрядной адресной шиной. Первый дешифратор адреса выполняет дешифрирование сигналов, поступающих по линиям A11, A12 и A13. На выходах 0–3 этого дешифратора формируются сигналы выбора модуля ПЗУ. Если на линии A13 устанавливается высокий уровень сигнала, то активируются линии 4–7 дешифратора адреса, что соответствует обращению к областям памяти с адресами, не меньшими чем 8192. Сигнал с любой из этих линий поступает на второй дешифратор адреса, который дешифрирует сигналы, поступающие по линиям A8 и A9. Второй дешифратор адреса используется для формирования сигнала выбора одного из блоков ОЗУ.

12. Первый таймер «делит» частоту 5 МГц (период 200 нс) на 16 383, поэтому на его выходе вырабатывается 1 импульс каждые 3,276600 мс. Следовательно, возможный диапазон интервалов времени между моментами появления сигналов на выходе второго таймера находится в пределах от 6,5532 мс до 53,680537 с.

13. а) Два формирователя: для линий D0–D3 и для линий D4–D7 шины данных. б) Другие линии не являются двунаправленными, и, следовательно, для них не требуется использовать шинные формирователи.

14. Это объясняется тем, что в ПЗУ нельзя записывать информацию во время работы системы.

15. Однокристальная микро-ЭВМ с ПЗУ используется в сравнительно недорогих системах, имеющих значительный объем программ. Микро-ЭВМ с ПЗУ с возможностью стирания применяется для сравнительно небольших систем, программы которых предполагается совершенствовать.

16. Вероятно, придется использовать микропроцессорную систему на нескольких кристаллах, поскольку в рассматриваемой системе контроля потребуется обеспечить ввод-вывод большого количества сигналов.

17. Выполняется пять измерений за один период. Частота сигнала равна 5 кГц, что соответствует 5000 период/с. С учетом работы восьми каналов оказывается, что преобразователь делает 200 000 измерений за 1 с.

18.в. 19.б. 20.а. 21.а. 22.в. 23.б. 24.г. 25.б. 26.б. 27.б.

Предметный указатель

Оглавление

Чарлз Гилмор

ВВЕДЕНИЕ В МИКРОПРОЦЕССОРНУЮ ТЕХНИКУ

Научный редактор Л. А. Паршина
Младший редактор Л. В. Тарасова
Художник В. Я. Медников
Художественный редактор В. Б Пришепа
Технические редакторы Н. И. Борисова, Г. Б. Алюлина, Н. И. Манохина
Корректор Н. В. Андреева

ИБ № 3731

Сдано в набор 06.12.83.
Подписано к печати 20.07.84.
Формат 70 × 100$^{1}/_{16}$ Объем 10,5 б л Бумага офсетная № 1.
Гарнитура таймс. Печать офсетная.
Усл. печ. л. 27,3. Усл. кр.-отт. 54,60.
Уч.-изд. л. 29,55. Изд. № 20/2787.
Тираж 50.000 экз. Зак. 1111. Цена 2 р. 50 к.

ИЗДАТЕЛЬСТВО «МИР»
129820, ГСП, Москва, И-110, 1-й Рижский пер., 2

Можайский полиграфкомбинат Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли.
143200, г. Можайск, ул. Мира, 93.