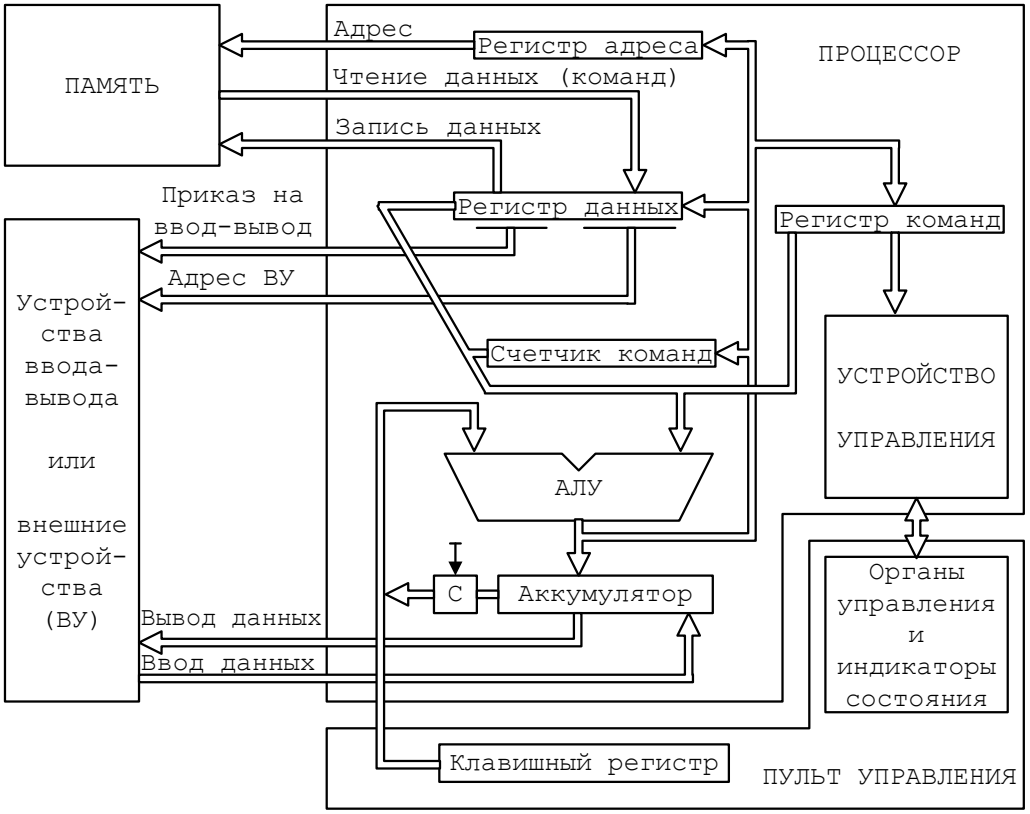
В.В. Кириллов

АРХИТЕКТУРА БАЗОВОЙ ЭВМ

Учебное пособие



Санкт-Петербург

2010

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

В.В. Кириллов

АРХИТЕКТУРА БАЗОВОЙ ЭВМ

Учебное пособие

Санкт-Петербург

2010

Кириллов В.В. Архитектура базовой ЭВМ
– СПб: СПбГУ ИТМО, 2010. – 144 с.

В учебном пособии рассматриваются основы функционирования и построения ЭВМ. Приводятся общие сведения о представлении и обработке информации в ЭВМ. Описывается функциональная модель гипотетической микроЭВМ (Базовой ЭВМ), структурно похожей на любые несложные ЭВМ, и работа с этой моделью. На ней можно исследовать взаимодействия устройств ЭВМ при выборке и исполнении команд и изучить ее функционирование на микропрограммном уровне. Эта ЭВМ впервые была подробно описана в [2,3] и с тех пор используется для обучения студентов на кафедре вычислительной техники, а также на многих кафедрах ИТМО и ряда других университетов. За 25 лет описания в [2,3] устарели и возникла необходимость их переработки и издания этого учебного пособия, в которое включены из [2] отредактированные разделы по Базовой ЭВМ.

Пособие будет использовано при подготовке бакалавров по направлениям 230100 «Информатика и вычислительная техника», 231000 «Программная инженерия» а также подготовки бакалавров других направлений, которым читают дисциплины, связанные с вычислительной техникой, как преподаватели кафедры ВТ, так и преподаватели других кафедр.

Рекомендовано Советом факультета КТиУ 13 октября 2009 г., протокол № 3

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена Программа развития государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на 2009–2018 годы.

ВВЕДЕНИЕ

В 1980-81гг на кафедре вычислительной техники (ВТ) Ленинградского института точной механики и оптики (ЛИТМО) проходили жаркие дискуссии о том, как обновить учебный класс по изучению Электронных вычислительных машин (ЭВМ). Стенды, созданные в 1960-70гг, морально и физически устарели и, кроме того, с их помощью нельзя было обучать студентов множества других кафедр вуза, которым кафедра ВТ читала дисциплины: «Вычислительная техника», «Вычислительная техника в инженерных и экономических расчетах» и т.п.

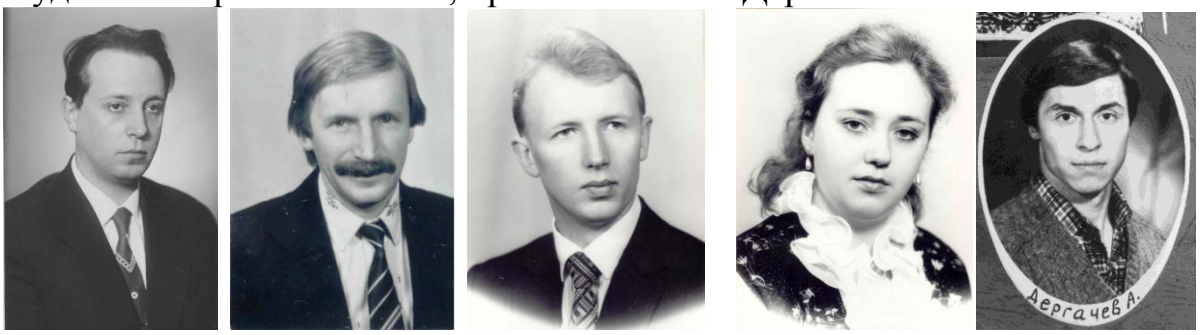
Предлагалось два пути:

1. создание учебных лабораторных комплексов на микропроцессорных комплектах
2. создание стендов для моделирования ЭВМ и любых цифровых устройств.

Первый путь позволял проводить физическое исследование микроЭВМ и цифровых устройств, но требовал периодической замены лабораторной базы. Так с 1983 года, когда были введены в строй учебные лабораторные комплексы на микропроцессорных комплектах К584, К589, К580, было создано несколько новых лабораторных комплексов и истрачено достаточно много средств. Кроме того, эти комплексы хороши для обучения специалистов, бакалавров и магистров по направлению подготовки «Информатика и вычислительная техника», но сложны и избыточны для обучения студентов других специальностей и направлений подготовки.

Второй путь предполагал создание устройства отображения информации, например такого, который описан в параграфе 5.2 и программной модели исследуемой ЭВМ или цифрового устройства, реализуемой на любой ЭВМ. Такой моделью могла стать (и стала) персональная ЭВМ. На ней легко изменять как структуру, так и программу изучения функционирования ЭВМ и (или) любого цифрового устройства.

В 1982-83 гг. доцент Кириллов В.В. разработал программную модель «Базовой ЭВМ», которая была реализована на "Искра-226" студентами Громовым Г.Ю., Громовой И.В. и Дергачевым А.М.

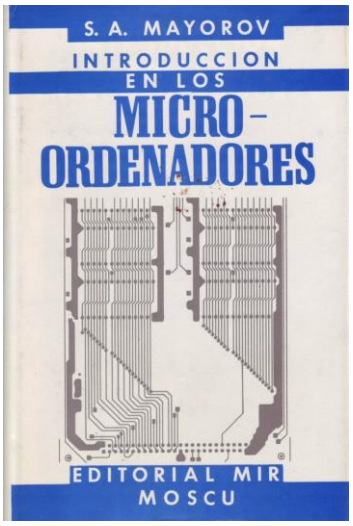


Доценты Кириллов В.В., Приблуда А.А. и студенты Громов Г.Ю., Громова И.В., Дергачев А.М.
во времена создания «Базовой ЭВМ»

В 1985 г. на кафедре был организован учебный класс на базе восьми ЭВМ "Искра-226", в котором проводились занятия по изучению базовой ЭВМ студентами института, преподавателями факультета повышения квалификации преподавателей и почти тысячью преподавателями вузов СССР, в рамках проводимой в 1985-86 гг. программы всеобщей компьютерной подготовки. Это послужило толчком к внедрению базовой ЭВМ в учебный процесс ряда вузов СССР (КПИ, ЛИИЖТ, ЛИСТ, ИМИ, ТПИ и др.).

В 1986 году был открыт новый класс для изучения базовой ЭВМ на, разработанных доцентом Приблуда А.А., специализированных стендах, управляемых микроЭВМ Электроника 60.

В 1988 году вышла книга «Введение в микроЭВМ» (переведена издательством «Мир» на испанский язык), где подробно описывалась эта модель. В нашем университете студенты до сих пор изучают на ней основы функционирования компьютеров, используя очередную версию, созданную для ПК. Эту версию разрабатывали выпускники кафедры (теперь её преподаватели) Гаврилов А.В. и Клименков С.В.



Предыдущие издания

Несмотря на закупку библиотекой университета в 1988 году нескольких сотен книг «Введение в микроЭВМ», за более чем двадцать лет она износилась и была частично утеряна читателями, из-за чего возникла необходимость издания тех разделов книги, которые непосредственно связаны с описанием базовой ЭВМ.

Созданным учебным пособием будут пользоваться как студенты первого курса направления (специальности) «Информатика и вычислительная техника», так и студенты более старших курсов других направлений и специальностей, которым читают дисциплины, связанные с вычислительной техникой, как преподаватели кафедры ВТ, так и преподаватели кафедр обеспечивающих подготовку по этим направлениям.

1. ОБЩИЕ СВЕДЕНИЯ О ПРЕДСТАВЛЕНИИ И ОБРАБОТКЕ ИНФОРМАЦИИ В ЭВМ

1.1. Две формы представления информации

Как известно, современные ЭВМ могут решать самые разнообразные задачи. Для этого лишь надо с помощью программы «научить» ЭВМ алгоритму решения той или иной задачи и ввести в нее исходные данные. Программа же записывается на алгоритмическом языке, который достаточно близок к естественному языку (особенно английскому).

Однако ЭВМ не понимает не только естественного языка, но и алгоритмического. Для расшифровки текста программы, написанной на алгоритмическом языке, в машине должна находиться специальная программа транслятор (от англ. translate – переводить), которая переводит текст исходной программы с алгоритмического языка на язык ЭВМ. А как же выглядит язык ЭВМ?

Вычислительная машина – это техническое устройство, в котором информация об исходных данных решаемой задачи, правилах ее решения (алгоритме), и результатах вычислений должна задаваться в виде изменения каких-либо физических величин:

углов поворота или перемещений (как, например, для «передачи информации» телевизору об уменьшении громкости или яркости);

намагниченности материала (как, например, для воспроизведения мелодии с помощью магнитофона или сохранения данных об аттестации студентов по дисциплинам учебного плана на магнитном диске);

освещенности экрана (дисплей) или фотодатчика (ввод информации с перфоленты или перфокарт).

В прошлые века, когда человечество не знало об электрических и магнитных явлениях или еще не умело их использовать, наиболее доступной, а следовательно, и удобной была механическая форма представления информации в вычислительных устройствах. В арифмометрах операции над числами выполнялись с помощью колес, которые при добавлении единицы поворачивались на 36° и с помощью штифта приводили в движение следующее по старшинству колесо всякий раз, когда цифра 9 переходила к цифре 0 (накапливался десяток). Однако механические устройства громоздки, дороги и слишком инерционны (с их помощью нельзя построить универсальные и быстродействующие вычислительные машины). Поэтому сейчас во всех вычислительных машинах в качестве основной формы представления информации служат электрические сигналы (чаще всего – уровни напряжения постоянного тока). Для передачи электрических сигналов нужны лишь провода, эти сигналы легко преобразовывать с помощью различных полупроводниковых схем.

При использовании в качестве носителя информации напряжений постоянного тока возможны две формы представления численного значения какой-либо переменной, например, X:

в виде одного сигнала – значения напряжения постоянного тока, которое сравнимо с величиной X (аналогично ей).

Например, при X=1845 единиц на вход вычислительного устройства можно подать напряжение 1,845 В (масштаб представления 0,001 В/ед.) или 9,225 В (масштаб представления 0,005 В/ед.);

в виде нескольких сигналов – нескольких значений напряжения постоянного тока, которые, например, сравнимы с числом единиц в X, числом десятков в X, числом сотен в X и т. д.

Первая форма представления информации называется аналоговой или непрерывной (с помощью сходной величины – аналога). Величины, представленные в такой форме, могут принимать принципиально любые значения в каком-то диапазоне. Они могут быть сколь угодно близки друг к другу, малоразличимы, но все-таки, хотя бы в принципе, различимы. Количество значений, которые может принимать такая величина, бесконечно велико. Их бесконечно много даже в случае, когда величина изменяется в ограниченном диапазоне, например 0–2000 или 0–0,0001. Отсюда названия – непрерывная величина и непрерывная информация. Слово непрерывность отчетливо выделяет основное свойство таких величин – отсутствие разрывов, промежутков между значениями, которые может принимать данная аналоговая величина.

Вторая форма представления информации называется цифровой или дискретной (с помощью набора напряжений, каждое из которых соответствует одной из цифр представляемой величины). Такие величины, принимающие не все возможные, а лишь вполне определенные значения, называются дискретными (прерывистыми). В отличие от непрерывной величины количество значений дискретной величины всегда будет конечным.

Сравнивая непрерывную и дискретную формы представления информации, нетрудно заметить, что при использовании непрерывной формы создателю вычислительной машины потребуется меньшее число устройств (каждая величина представляется одним, а не несколькими сигналами), но эти устройства будут сложнее (они должны различать значительно большее число состояний сигнала). Кроме того, отметим, что устройства для обработки непрерывных сигналов обладают более высокой «квалификацией» (они могут интегрировать сигнал, выполнять любое его функциональное преобразование и т. п.) и за счет этого, а также ряда других особенностей имеют высокое быстродействие. (Некоторые виды задач решаются во много раз быстрее, чем с помощью устройств с дискретным представлением информации.)

Однако из-за сложности технической реализации устройств для

логических операций с непрерывными сигналами, длительного хранения таких сигналов, их точного измерения подобная форма представления в основном используется в аналоговых вычислительных машинах (АВМ). Эти машины предназначены для решения задач:

- описываемых системами дифференциальных уравнений:
- исследования поведения подвижных объектов (машин, роботов, судов, летательных аппаратов и т. п.);
- моделирования ядерных реакторов, гидротехнических сооружений, газовых сетей, электромагнитных полей и биологических систем;
- решения задач параметрической оптимизации и оптимального управления; управления процессами переработки нефти и выплавки стали.

Но АВМ не могут решать задачи, связанные с хранением и обработкой больших объемов информации, которые легко решаются при использовании цифровой (дискретной) формы представления информации, реализуемой цифровыми электронными вычислительными машинами (ЭВМ).

Резюме:

1. Исходные данные, результаты и другая информация, перерабатываемая вычислительными машинами, представляется в них в виде каких-либо физических величин – чаще всего электрических сигналов (напряжений постоянного тока).

2. Существуют две формы представления информации (физических величин) в вычислительных машинах: аналоговая (непрерывная) и цифровая (дискретная). В первой величина представляется в виде одного сигнала, пропорционального этой величине, во второй – в виде нескольких сигналов, каждый из которых соответствует одной из цифр заданной величины.

3. Непрерывная форма используется в электронных аналоговых вычислительных машинах, а дискретная – в цифровых электронных вычислительных машинах. ЭВМ – универсальные машины, позволяющие решать любые задачи науки и техники (круг таких задач все время увеличивается). Однако те задачи, где не требуется хранить и обрабатывать большие объемы информации, которые описываются системами дифференциальных уравнений и должны решаться за минимальное время, целесообразнее решать с помощью АВМ. Создают также и гибридные вычислительные системы (ГВС), которые соединяют в себе достоинства как аналоговых, так и цифровых вычислительных машин.

1.2. Способы представления дискретной информации

Как было показано выше, каждое значение из набора исходных данных задачи и (или) результатов ее решения может быть представлено в ЭВМ в виде нескольких электрических сигналов. Один из этих сигналов

соответствует числу единиц в значении, другой – числу десятков, третий – числу сотен и т. д. Однако такое естественное для нас представление не является наилучшим с технических позиций. Устройство, предназначенное для обработки подобных сигналов, должно различать в каждом из них десять состояний. Значительно проще построить устройство, которое различало бы всего два состояния сигнала (его наличие или отсутствие). Это тем более целесообразно, так как существующие сейчас дешевые устройства для ввода данных в ЭВМ также кодируют (представляют) отдельные составляющие вводимой информации с помощью двух состояний:

- отверстие есть или отсутствует (перфолента и перфокарта);
- материал намагничен или размагничен (магнитные ленты, диски);
- тумблер включен или выключен (отдельные блоки пульта управления ЭВМ);
- кнопка нажата или нет (клавиатура дисплеев).

Это обстоятельство натолкнуло создателей первых ЭВМ на применение другой системы счисления при внутреннем представлении чисел в машинах: вместо привычной десятичной системы, которая использовалась в механических вычислительных устройствах, была взята двоичная система счисления. Как и десятичная система счисления, двоичная система (в которой используются лишь цифры 0 и 1) является позиционной системой счисления, т. е. в ней значение каждой цифры числа зависит от положения (позиции) этой цифры в записи числа. Каждой из позиций присваивается определенный вес. Так, число 371 можно записать в виде:

$$3*10^2 + 7*10^1 + 1*10^0 = (371),$$
$$300 + 70 + 1,$$

где цифры имеют вес 10^1,
или в двоичной системе счисления:

$$256 + 0 + 64 + 32 + 16 + 0 + 0 + 2 + 1 =$$
$$= 1*2^8 + 0*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 =$$
$$= (101110011)_2 = (371)_{10},$$

где цифры имеют вес 2^1.

Здесь и далее основание системы счисления указано в виде индекса для числа, взятого в круглые скобки,

Существуют специальные термины, широко используемые в вычислительной технике: бит, байт и слово.

Двоичный разряд обычно называют битом. Таким образом, число 1001 является 4-битовым двоичным числом, а ранее рассмотренный двоичный эквивалент числа 371, т. е. 101110011, – 9-битовым числом. Крайний слева бит числа называется старшим разрядом (он имеет наибольший вес), крайний справа – младшим разрядом (он имеет наименьший вес). Двоичное число, состоящее из 16 бит, представлено на

рис. 1.1.

Эволюция вычислительной и информационной техники вызвала появление 8-битовой единицы для обмена информацией между устройствами. Такая 8-битовая единица носит название байта. Многие новые типы ЭВМ и дискретных систем управления перерабатывают информацию порциями (словами) по 8, 16, 32 или 64 бита (1, 2, 4 и 8 байт). Двоичное число, состоящее из 2 байт, показано на рис. 1.1.

ЭВМ содержит большое количество ячеек памяти и регистров (от лат. regestum – внесенное, записанное) для хранения двоичной информации. Большинство этих ячеек имеет одинаковую длину *n,* т. е. они используются для хранения *n* бит двоичной информации. Информация, хранимая в такой ячейке, называется словом. Слово 16-битовой ЭВМ представлено на рис. 1.1.

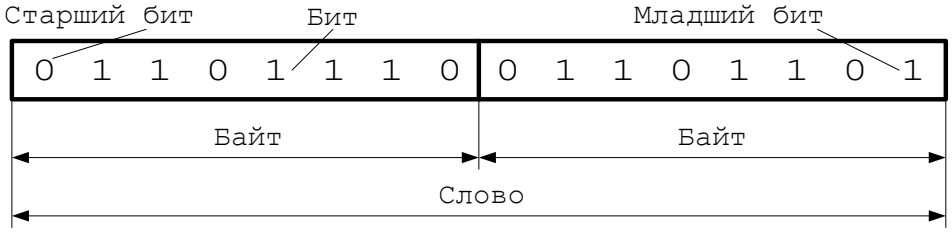


Рис. 1.1. Бит, байт и слово

Ячейки памяти и регистры состоят из элементов памяти. Каждый из таких электрических элементов может быть в одном из двух устойчивых состояний:

- конденсатор заряжен или разряжен,
- транзистор находится в проводящем или непроводящем состоянии,
- специальный полупроводниковый материал имеет высокое или низкое удельное сопротивление и т. п.

Одно из таких физических состояний создает высокий уровень выходного напряжения элемента памяти, а другое – низкий. В элементах памяти ряда микроЭВМ это электрические напряжения порядка 4 В и 0 соответственно, причем первое обычно принимается за двоичную единицу, а второе – за двоичный нуль. (Хотя возможно и обратное кодирование.)

На рис. 1.2 показан выходной сигнал такого элемента памяти (например, одного разряда регистра) при изменении его состояний (при переключениях) под воздействием некоторого входного сигнала. Хотя переход от 0 к 1 и от 1 к 0 происходит не мгновенно, однако в определенные моменты времени этот сигнал достигает значений, которые воспринимаются элементами ЭВМ как 0 или 1.

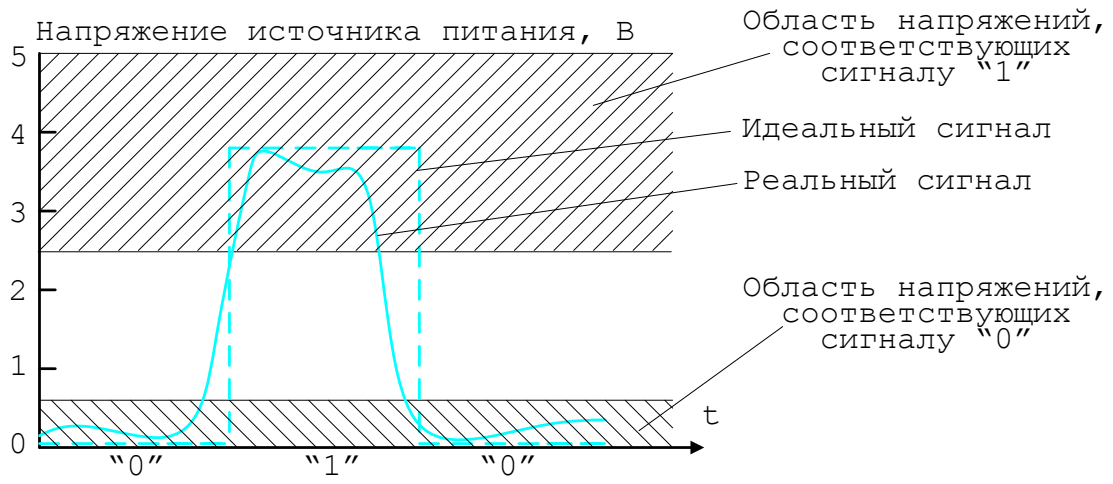


Рис. 1.2. Графическое изображение двоичного сигнала

Регистры часто изображают так, как показано на рис. 1.3,*а*. Регистр характеризуется единственным числом: количеством битов, которые могут в нем храниться. На рис. 1.3,*а* разряды (биты) 16-битового регистра пронумерованы степенями двойки, соответствующими позициям битов двоичного числа, находящегося в регистре. Помещенная в регистр информация остается там до тех пор, пока она не будет заменена другой. Процесс чтения информации из регистра не влияет на содержимое последнего. Говоря другими словами, операция чтения информации, хранимой в регистре, сводится к созданию копии его содержимого, оригинал же сохраняется в регистре без изменений.

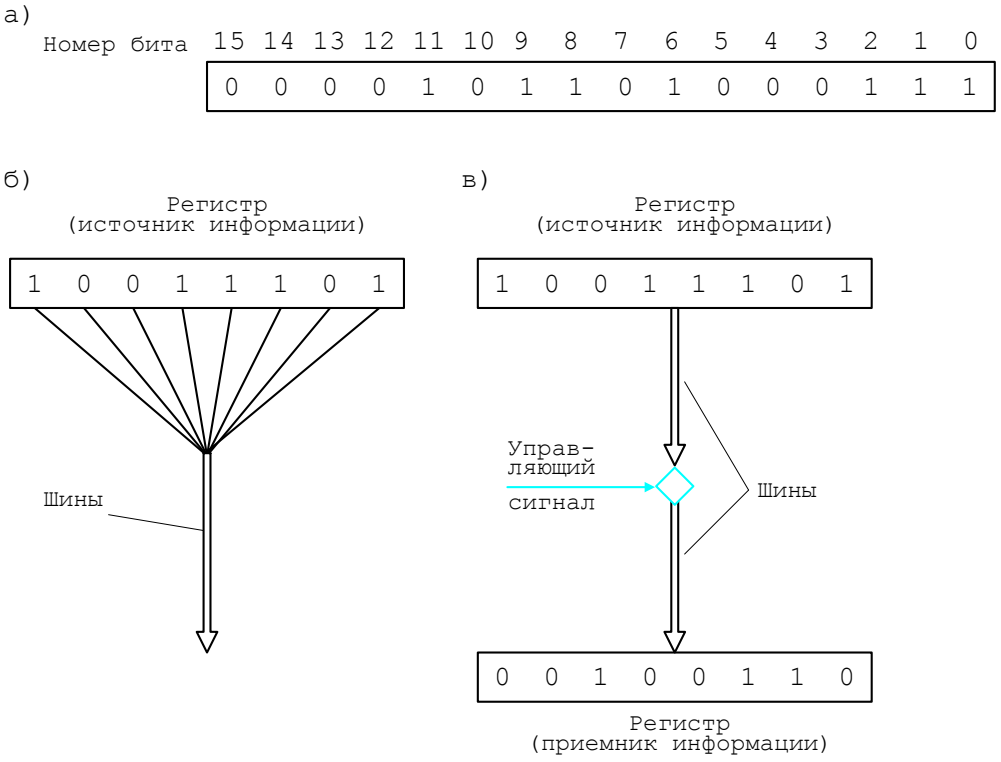


Рис. 1.3. Регистры, шины и вентильные схемы

Электрическая цепь, соединяющая регистр с другим регистром или иным устройством ЭВМ, называется шиной (bus). Шина состоит из параллельных проводов, каждый из которых предназначен для передачи соответствующего бита регистра (рис. 1.3, *б)*. Два 8-битовых регистра соединяются между собой шиной из восьми проводов. Про такую шину говорят, что ее ширина равна 8. В действительности шина обычно содержит несколько дополнительных проводов, используемых для передачи сигналов синхронизации и управления, однако подобный анализ структуры шин нас пока не интересует.

Несмотря на то, что электрический провод позволяет передавать сигналы в любом направлении, шина служит для передачи информации лишь в направлении, обозначенном стрелкой на шине (рис. 1.3, *б, в)*. Такая однонаправленность передачи обусловлена не свойствами шины, а характеристиками схем, соединяющих шину с регистром или другими устройствами ЭВМ. Специальные схемы позволяют, например, в одни моменты времени передавать информацию по шине в одну сторону, а в другие – в обратном направлении, т. е. организовать двунаправленную шину.

Вентильные схемы – это электронные ключевые схемы, предназначенные для управления потоком информации из регистров в шины и обратно. На рис. 1.3, *в* показано применение вентильной схемы *B*. Такая схема имеет два входа и один выход. На один вход подается информационный сигнал (данные с регистра), а на другой – управляющий, Если управляющий сигнал равен единице, то данные проходят через схему без препятствий, как будто ее и нет. Если управляющий сигнал равен нулю, никакая информация не пройдет через схему. Для подачи информационного сигнала на вход вентильной схемы обычно используется многопроводная шина. Для передачи выходного сигнала вентильной схемы требуется шина с таким же количеством проводов. Если управляющий сигнал равен единице, то информационные сигналы на входной и выходной шинах совпадают. Таким образом, при подаче единичного управляющего сигнала на вентильную схему *B* состояние регистра приемника информации изменится с $(00100110)_2$ на $(10011101)_2$, т. е. в него поступит копия содержимого регистра источника информации.

Тактовые импульсы (вырабатываемые генератором тактовых импульсов ЭВМ) используются для синхронизации процессов передачи информации между устройствами ЭВМ. Поскольку на простом примере трудно проиллюстрировать необходимость распределения во времени отдельных операций по передаче информации в цифровых устройствах, потратим немного времени на рассмотрение работы упрощенной модели школьного микрокалькулятора (рис. 1.4, *а)*.

В состав микрокалькулятора входят:

- цифровая (0, 1, ...,9,«.») и функциональная (C, %, ×, ÷, –, √, +, =)

клавиатуры;

- основной регистр X, в который вводится операнд (набираемое на клавиатуре число) и по окончании какой-либо операции засылается ее результат (сумма, разность и т. п.);
- табло для индикации содержимого регистра X, т. е. вводимого числа (в момент набора его на цифровой клавиатуре) или результата вычислений (после нажатия функциональной клавиши);
- вспомогательный регистр Y, в который переписывается содержимое регистра X при вводе в последний нового операнда, что позволяет, например, сохранить в Y значение предыдущего слагаемого или ранее накопленной суммы в момент ввода в X нового слагаемого;
- арифметико-логическое устройство (АЛУ), предназначенное для выполнения операций над содержимым регистров X и Y (сложение, умножение, вычитание и деление) или только регистра X (извлечение квадратного корня), а также ряда служебных операций (например, сдвига старших цифр числа при вводе его новой цифры);
- устройство управления вводом числа и выполнением операций, которое после любого нажатия цифровой или функциональной клавиши вырабатывает определенную последовательность управляющих сигналов ($У_i$) для осуществления необходимых пересылок информации между устройствами микрокалькулятора и (или) перестройки АЛУ на ту или иную операцию.

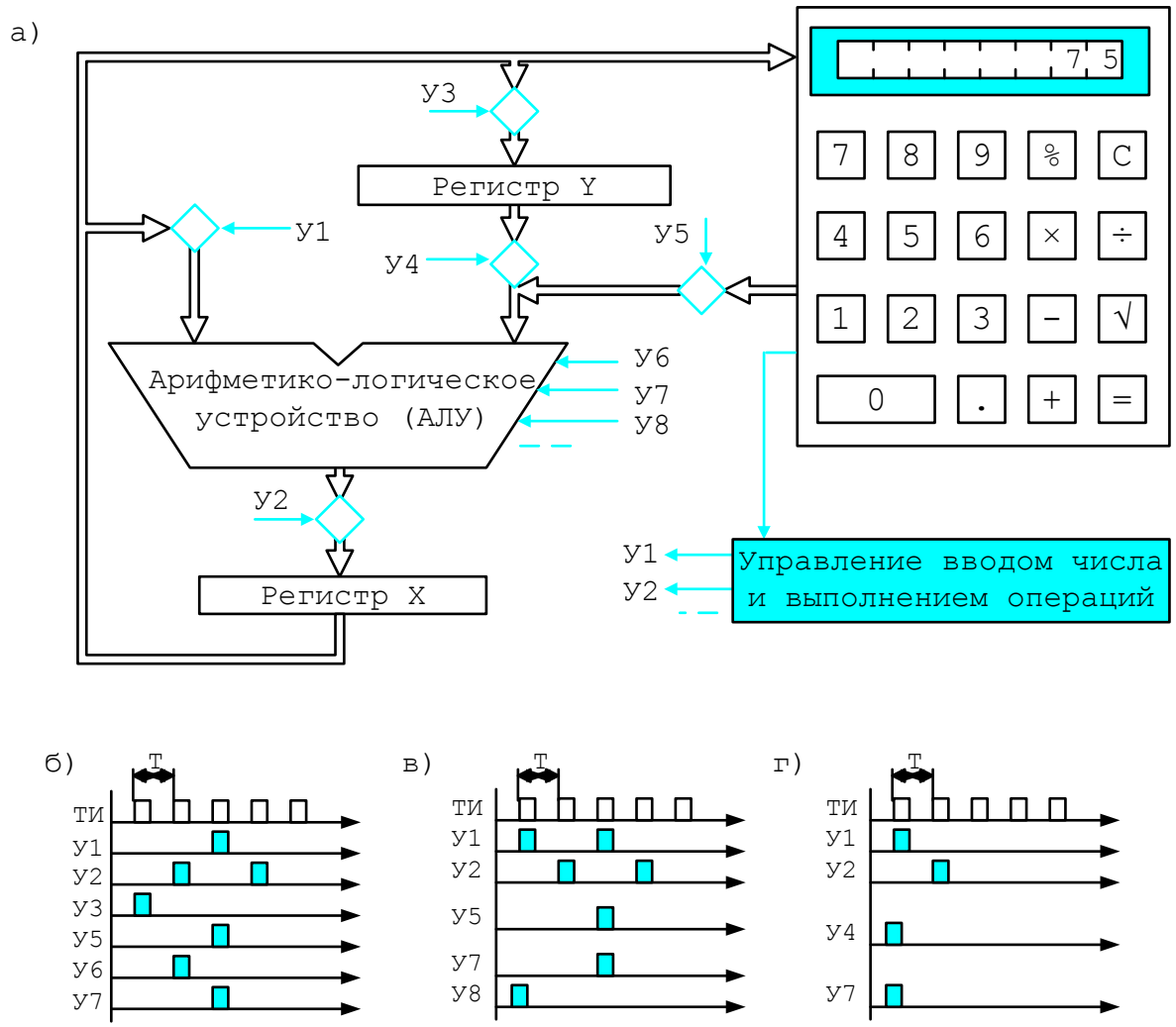


Рис. 1.4. К необходимости синхронизации процессов передачи информации между устройствами ЭВМ

Кроме того, отметим, что хотя микрокалькулятор работает с сигналами, которые кодируются двоичными цифрами 0 и 1, в примере будет использовано знакомое десятичное представление цифр в регистрах. Это оправдывается еще и тем, что числа, обрабатываемые в микрокалькуляторах, кодируются в так называемой двоично-десятичной системе счисления.

Что будет происходить с регистрами микрокалькулятора в процессе ввода слагаемых и получения суммы чисел 27 + 48+12976 + 38+..,? Начнем с момента нажатия клавиши « + », завершающей ввод первого слагаемого, т. е. с момента, когда в регистрах содержатся следующие числа:

Регистр Y	0
Регистр X	27

1-й шаг. Ввод цифры 4 – старшей цифры второго слагаемого:

Регистр Y	27
Регистр X	4

Содержимое регистра X переписалось для сохранения в регистр Y,

регистр X был очищен и в его младший разряд записан код нажатой клавиши, т. е. цифра 4.

2-й шаг. Ввод цифры 8 – младшей цифры второго слагаемого:

Регистр Y	27
Регистр X	48

Число, содержащееся в регистре *X,* было сдвинуто влево на один разряд, а в освободившийся младший разряд записан код нажатой клавиши, т. е. цифра 8.

3-й шаг. Нажатие клавиши «+»:

Регистр Y	27
Регистр X	75

Зафиксировался ввод последней цифры второго слагаемого (48), выполнились операция сложения содержимого регистров X и Y (48 + 27 = 75) и операция пересылки полученного результата в регистр X.

4-й шаг. Ввод цифры 1 – старшей цифры третьего слагаемого:

Регистр Y	75
Регистр X	1

Содержимое регистра X переписалось... (см. комментарии к 1-му шагу).

Дальнейшие шаги не представляют интереса, так как. в них будут повторяться ранее описанные операции. Поэтому перейдем к рассмотрению последовательности управляющих сигналов $У_i$, вырабатываемых устройством управления при выполнении трех первых шагов.

Ввод первой цифры любого операнда (первое нажатие цифровой клавиши после выполнения какой-либо операции) должен инициировать перепись содержимого регистра X в регистр Y, т. е. выработку управляющего сигнала $У_3$ (рис. 1.4,6). Одновременно с этим сигналом может быть выработан сигнал $У_6$, устанавливающий 0 на выходе АЛУ. Теперь производятся очистка регистра X, суммирование его содержимого с вводимой цифрой и перепись полученной суммы в регистр X. Для этого сначала подается сигнал $У_2$ (0 с выхода АЛУ переписывается в регистр X), затем одновременно сигналы $У_7$, $У_1$ и $У_5$ ($У_7$ для настройки АЛУ на суммирование) и, наконец, сигнал $У_2$.

Ввод второй (или любой из следующих цифр) осуществляется путем умножения на 10 (сдвига влево на один десятичный разряд) содержимого регистра X и суммирования сдвинутого значения с вводимой цифрой. Для этого вырабатывается последовательность управляющих сигналов, показанная на рис. 1.4,е (сигнал $У_8$ служит для организации сдвига).

Суммирование содержимого регистров X и Y (сигналы $У_7$, $У_1$ и $У_4$) и копирование полученной суммы в регистр X (сигнал *У2)* иллюстрируются временной диаграммой на рис. 1.4, *г.*

Обсудим, наконец, почему некоторые из управляющих сигналов

временных диаграмм на рис. 1.4, *б, в, г* вырабатываются одновременно, а другие – через те или иные интервалы времени. Например, можно ли при суммировании содержимого регистров X и Y одновременно вырабатывать сигналы $У_1$, $У_2$, $У_4$ $У_7$? Простейший анализ позволяет установить, что при одновременной выработке сигналов $У_1$ и $У_2$, а также при настройке АЛУ на выполнение какой-либо операции с содержимым регистра X возникает полная неопределенность: содержимое регистра X попадает в АЛУ, через него обратно в регистр X и т. д. С другой стороны, если при суммировании X и Y не будет обеспечена одновременная выработка сигналов $У_1$, $У_4$ и $У_7$, то суммы не получится.

А чем же определяется минимальный промежуток времени между управляющими сигналами, которые должны быть сдвинуты относительно друг друга (например, промежуток времени между сигналами $У_1$ и $У_2$ на рис. 1.4,*г)*? Этот промежуток времени T выбирается исходя из максимального времени переключения элементов ЭВМ (пока, например, не окончилось формирование суммы X+Y, бессмысленно подавать управляющий сигнал $У_2$ на перепись этой суммы в регистр X). Для выдерживания нужных промежутков времени в управляющих устройствах ЭВМ устанавливаются генераторы тактовых импульсов (ГТИ), синхронизирующие управляющие сигналы $У_i$.

Из приведенных выше примеров видно, что с помощью электрических сигналов можно достаточно просто представлять двоичные цифры. Эти сигналы (цифры) легко пересылать между устройствами ЭВМ. Они могут изменять состояния устройств для хранения цифр (элементов памяти регистров и ячеек памяти) и т. д. Однако во всех примерах рассматривались лишь двоичные эквиваленты целых десятичных чисел без знака и могло сложиться впечатление, что 0 и 1 нельзя использовать для кодирования и обработки другой информации. Это не так, наборы двоичных цифр (так же, как наборы точек и тире в азбуке Морзе) позволяют закодировать любую информацию (вплоть до графической и звуковой).

Если надо закодировать целое число со знаком, то старший бит регистра (ячейки памяти) используется для хранения знака (например, 0 при положительном знаке числа и 1 при отрицательном). На рис. 1.5,*а* показано такое представление числа со знаком для 16-битовой ЭВМ. Числа вида $\pm M \cdot 10^{\pm p}$ (например, $384 \cdot 10^{-3}$ или $-9 \cdot 10^{-5}$) могут кодироваться так, как показано на рис. 1.5, б.

Однако в 16 разрядах нельзя сохранить хоть сколько-нибудь приемлемое число цифр мантиссы и чаще всего для кодирования подобных чисел используются слова длиной от 32 до 64 бит. Буквы русского и латинского алфавитов, знаки операций и другие символы кодируются с помощью 8-ми или 16-битовых (1 или 2-байтовых) чисел (рис. 1.5, в), что позволяет закодировать $2^8 = 256$ различных символов (см. табл. Б.1 и Б.2

приложения Б).

Аналогичным образом (в виде двоичных чисел) кодируются команды – приказы на выполнение каких-либо операций со словами информации. Так, в четырех старших битах команды на рис. 1.5,*г* может быть закодирована операция (например, вычитание), далее адреса ячеек памяти, из которых надо взять операнды (уменьшаемое и вычитаемое), и, наконец, адрес ячейки памяти, в которую помещается результат выполненной операции (разность).

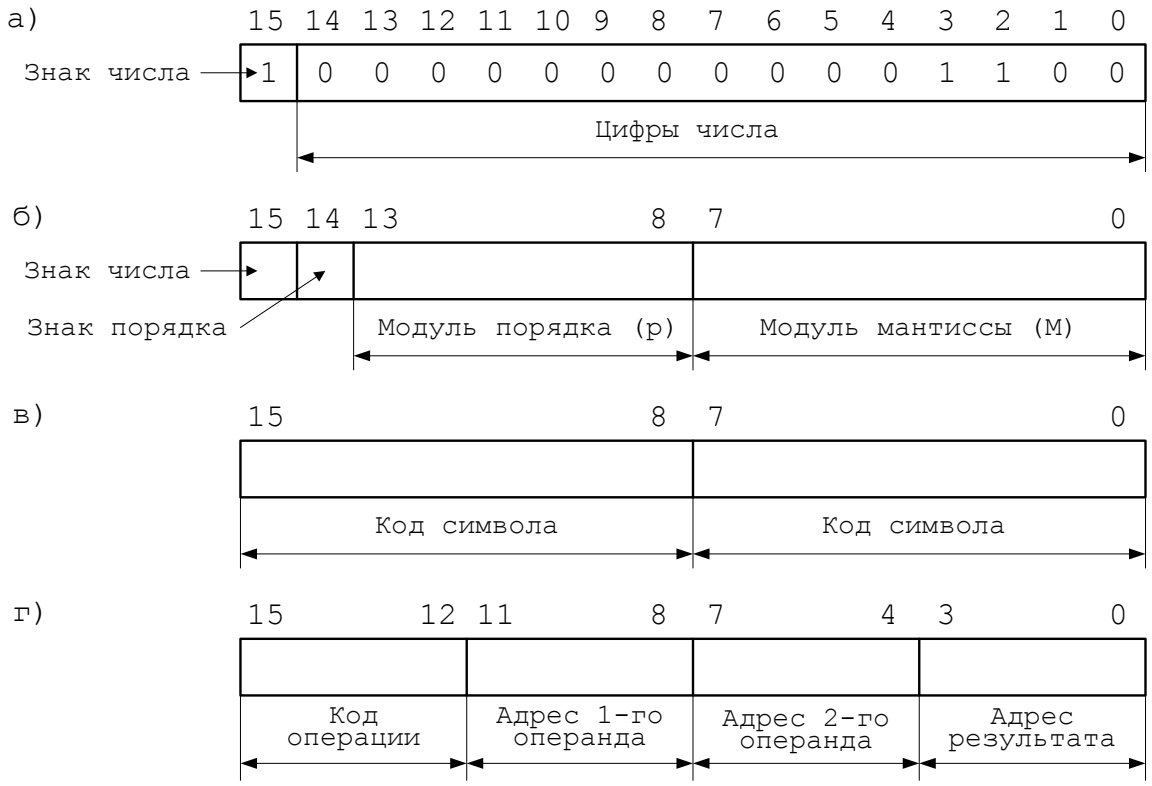


Рис. 1.5. Форматы представления информации в 16-битовой ЭВМ:

а – целые числа; б – числа с плавающей запятой; *в* – символы; *г* – команды

Когда нам приходится сталкиваться с информацией вида 50 369, -15 167, «Да» или «С4С1; перейти к команде, расположенной в ячейке памяти с адресом $(4C1)_{16}$», мы легко различаем перечисленные объекты по их внешнему виду. На первом месте расположено целое число без знака, на втором – целое число со знаком, на третьем – русское слово «Да» (так как использован символ русского алфавита – «Д»), а на четвертом – команда безусловного перехода базовой ЭВМ с пояснением к ней.

Однако без пояснений нельзя понять, какую информацию несет набор битов 1100 0100 1100 0001, с помощью которого кодируются в ЭВМ все перечисленные выше объекты. (В этом случае число -15 167 закодировано в так называемом дополнительном коде, о котором рассказано в параграфе 2.4.)

А как же различает их ЭВМ? Есть два пути, позволяющие

информировать ЭВМ о том, что закодировано в принимаемой ею последовательности битов.

Первый путь связан с дополнением последовательности рядом идентифицирующих битов – метабитов. Так, вводя в последовательность один дополнительный метабит и записывая в него, например, 0 для команд и 1 для других видов информации, можно легко научить ЭВМ отличать команды от данных. Несколько метабитов помогли бы однозначно определять символы, целые числа и т. д. Однако при таком способе идентификации последовательности битов требуется увеличивать объем памяти ЭВМ для хранения метабитов, усложнять ее структуру за счет введения схем анализа метабитов, т. е. необходимы дополнительные затраты как материальных ресурсов, так и времени (на анализ метабитов).

Второй путь, используемый при работе с большинством современных ЭВМ, не связан с затратой материальных или временных ресурсов. В нем от программиста требуются детальное распределение в памяти ЭВМ команд программы и данных, точное указание в командах адресов операндов (результата или перехода) и осуществление ряда других мероприятий, исключающих возможность применения операнда в качестве команды, команды – в качестве операнда и внесения прочих ошибок, которые не могут быть выявлены в процессе выполнения программы. Труд программиста несколько облегчается при использовании им для написания программ какого-либо алгоритмического языка, но и в этом случае приходиться детально описывать типы переменных, размерности массивов и т, п.

Резюме

1. Конструкция ЭВМ предельно упрощается и ЭВМ работает наиболее надежно (устойчиво), если сигналы, циркулирующие в электронных схемах ЭВМ, используются для представления только двух значений: 0 и 1.

2. Бит (от англ. binary digit) –двоичная цифра. Бит может принимать лишь одно из двух значений – 0 или 1 (да или нет, включено или выключено).

3. Группа из восьми соседних разрядов (битов), с которой ЭВМ оперирует как с одним целым при передаче, хранении и обработке информации, называется байтом. Байт обычно используется для представления либо букв и специальных символов, либо пары десятичных или шестнадцатеричных цифр.

Более крупные единицы информации – слова. Они обычно кратны по длине байту, и это значительно упрощает согласование процессов обработки информации в ЭВМ.

4. Двоичные цифры представляются в ЭВМ сигналами (чаще всего напряжениями), четко различаемыми элементами машины. Хотя эти

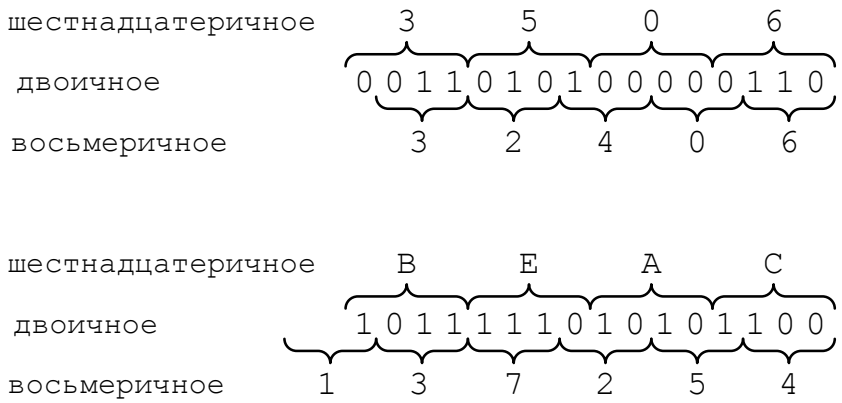
сигналы непрерывны, смысл двоичных символов (0 или 1) они имеют в определенные дискретные моменты времени, которые чаще всего задаются с помощью тактирующих импульсов.

5. С помощью последовательности двоичных цифр можно закодировать, хранить и обрабатывать в ЭВМ любую информацию (числа, тексты, команды и т. п.). Разнотипные слова информации обычно отличаются друг от друга способом использования, но не способами кодирования.

1.3. Системы счисления, используемые в вычислительной технике

Удобная для использования в ЭВМ двоичная система счисления совсем неудобна для записи и чтения чисел человеком. Действительно, вместо, например, четырехзначного десятичного числа $(8769)_{10}$ приходится работать с его 14-разрядным эквивалентом – двоичным числом $(10001001000001)_2$. Правда, к этому прибегают лишь при общении с ЭВМ на уровне ее машинного языка. Однако в настоящее время такой язык часто используется при работе с микропроцессорами и микроЭВМ.

Для сокращения трудоемкости ручной обработки кодов чисел, команд широко применяют восьмеричную и шестнадцатеричную системы счисления. В восьмеричной системе счисления используются 8 цифр (0, 1, 2, 3, 4, 5, 6, 7), в шестнадцатеричной – 10 цифр и 6 прописных латинских букв от A до F (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Так как основанием восьмеричной системы является число $8 = 2^3$, то для перевода двоичных чисел в восьмеричные необходимо разделить двоичные числа на 3-битовые группы – триады. Каждую из таких групп можно представить одной восьмеричной цифрой, используя табл. А.1 эквивалентности кодов (см. приложение А). Аналогичным образом осуществляется перевод двоичных чисел в шестнадцатеричные ($16 = 2^4$). Только в этом случае двоичное число разбивается на 4-битовые группы (тетрады), которые и представляются одной шестнадцатеричной цифрой. Покажем два примера подобных преобразований:



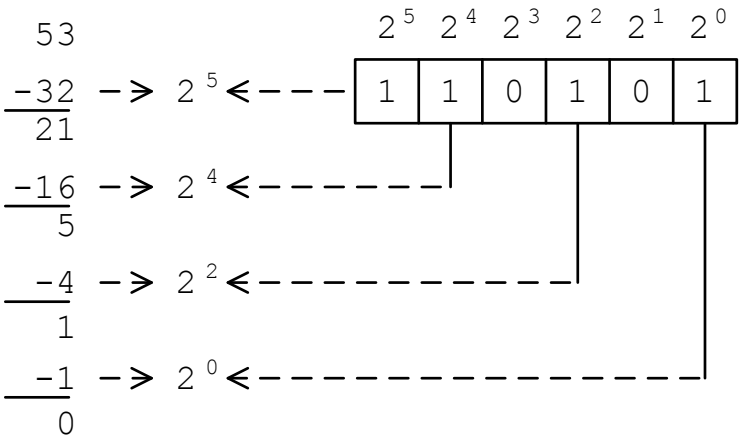
Преобразование двоичных чисел в десятичные может быть выполнено различными способами. Простейший из них вытекает непосредственно из определения двоичных чисел. Преобразование осуществляется путем суммирования значений степеней числа 2, соответствующих тем разрядам переводимого двоичного числа, в которых содержатся единицы. Например, переведем в десятичную систему число $(10111010)_2$:

7	6	5	4	3	2	1	0	разряды переводимого числа
1	0	1	1	1	0	1	0	переводимое число
2_7	2_6	2_5	2_4	2_3	2_2	2_1	2_0	степень числа 2

128+0+32+16+8 +0 +2 +0 =186 результат перевода

Следовательно, $(10111010)_2 = (186)_{10}$.

Обратное преобразование можно выполнить, в частности, следующим образом. Из переводимого десятичного числа вычитается наибольшее значение степени числа 2, не превышающее заданного числа. Затем операция повторяется для полученной разности (остатка). Как только заданное число окажется полностью разложенным на значения степеней 2, его искомое двоичное выражение можно скомпоновать из единиц в битовых позициях, соответствующих имеющимся в разложении степеням 2, и нулей во всех прочих позициях. Например:



Следовательно, $(53)_{10} = (110101)_2$.

Значения степеней числа 2, которые используются для указанных переводов, приведены в табл. А.2 приложения А. Первая строка этой таблицы позволяет получить степени числа 16 (16^0, 16^1, 16^2, 16^3, 16^4).

Аналогичным образом можно выполнить перевод восьмеричных и шестнадцатеричных чисел в десятичные и наоборот. При переводе двоичных и восьмеричных чисел в шестнадцатеричные рекомендуем использовать таблицы приложения А.

Арифметические операции с двоичными, восьмеричными и шестнадцатеричными числами осуществляются по тем же правилам, что и

с десятичными числами, за исключением того, что переносы в следующие разряды производятся при достижении 2, 8 и 16, а не 10 как в десятичной системе. Подробно ряд арифметических операций рассмотрим в параграфе 2.4, а здесь приведем пример сложения чисел $(166)_{10}$ и $(47)_{10}$ в двоичной, восьмеричной и шестнадцатеричной системах счисления:

Перенос	1 111	11	1
1-е слагаемое	10100110	246	A6
2-е слагаемое	00101111	57	2F
Результат	11010101	325	D5

Наконец, следует упомянуть и о двоично-десятичной системе счисления, которая широко используется в цифровых устройствах, где основная часть операций связана не с обработкой и хранением вводимой информации, а с самим ее вводом, и выводом на какие-либо индикаторы с десятичным представлением полученных результатов (микрокалькуляторы, кассовые аппараты и т. п.).

В двоично-десятичной системе десятичные цифры от 0 до 9 представляют 4-разрядными двоичными комбинациями от 0000 до 1001, т. е. двоичными эквивалентами десяти первых шестнадцатеричных цифр (см. табл. А.1 приложения А). Преобразования из двоично-десятичной системы в десятичную (и обратные преобразования) не вызывают затруднений и выполняются путем прямой замены четырех двоичных цифр одной десятичной цифрой (или обратной замены). Две двоично-десятичные цифры составляют 1 байт. Таким образом, с помощью 1 байта можно представлять значения от 0 до 99, а не от 0 до 255 или от 0 до FF, как при использовании 8-разрядного двоичного числа или 2-разрядного шестнадцатеричного числа. Используя 1 байт для представления каждых двух десятичных цифр, можно формировать двоично-десятичные числа с любым требуемым числом десятичных разрядов.

Так, если число

$$1001\ 0101\ 0011\ \ 1000$$

рассматривать как двоичное, то его десятичный эквивалент

$$(1001\ \ 0101\ 0011\ \ \ 1000)_{2} = (38200)_{10}$$

в 4 раза больше десятичного эквивалента двоично-десятичного числа

$$(1001\ \ 0101\ \ 0011\ 1000)_{2–10} = (9538)_{10}.$$

Сложение двоично-десятичных чисел, имеющих один десятичный разряд, выполняется так же, как и сложение 4-разрядных двоичных чисел без знака, за исключением того, что при получении результата, превышающего 1001, необходимо производить коррекцию. Результат корректируется путем прибавления двоичного кода числа 6, т. е. кода 0110.

Например:

```
  4    0100      5    0101                    9    1001
+ 5  + 0101    + 9  + 1001                  + 9  + 1001
  9    1001      14   1110                    18   1 0010
                  + 0110 – коррекция              + 0110 - коррекция
               10+4 1 0100                   10+8 1 1000
```

Если первоначальное двоичное сложение или прибавление корректирующего числа приводит к возникновению переноса, то при сложении многоразрядных двоично-десятичных чисел осуществляется перенос в следующий десятичный разряд:

```
  1889      0001      1000      1000      1001
+ 6376    + 0110    + 0011    + 0111    + 0110
  8265      0111      1011      1111      1111
                    + 0110    + 0110    + 0110
                              +     1     0101
                      1         0110
                    + 1
                      0010
              + 1
                1000
```

Резюме

1. Для удобства выполнения ручных операций по подготовке и отладке программ для микроЭВМ двоичные коды команд и данных обычно заменяют на восьмеричные или шестнадцатеричные.

2. Перевод двоичных чисел в восьмеричные (шестнадцатеричные) выполняется путем разбиения этого числа на 3-битовые (4-битовые) группы и замены каждой из групп на восьмеричную (шестнадцатеричную) цифру.

3. Перевод восьмеричных (шестнадцатеричных) чисел в двоичные осуществляется путем замены каждой восьмеричной (шестнадцатеричной) цифры на ее 3-битовый (4-битовый) двоичный код.

4. Двоичное число можно преобразовать в десятичное, представив его в виде суммы степеней числа 2.

5. Десятичное число можно преобразовать в двоичное путем последовательного вычитания степеней числа 2, начиная с наибольшей степени, содержащейся в числе. Найденные в результате коэффициенты располагают в последовательности таким образом, чтобы коэффициент при старшей степени числа 2 стоял в старшем разряде двоичного числа.

6. Ряд микроЭВМ может обрабатывать информацию, представленную в виде двоичных кодов десятичных цифр, т. е. как бы в десятичной системе счисления. Подобная система счисления получила название двоично-десятичной. Перевод чисел из десятичной системы

счисления в двоично-десятичную (и обратное преобразование) выполняется путем прямой замены каждой десятичной цифры числа на четыре двоичные (или обратной замены).

7. Основание системы счисления часто указывают в виде индекса для числа, взятого в круглые скобки.

1.4. Структура и принцип функционирования ЭВМ

Типичная ЭВМ состоит из процессора, памяти и устройств ввода-вывода (рис. 1.6). Со времени появления в 40-х годах первых электронных цифровых вычислительных машин технология производства каждой из этих трех подсистем была значительно усовершенствована. За последнее десятилетие благодаря развитию интегральной технологии существенно улучшились характеристики процессоров и памяти. Кроме того, была снижена их стоимость. В настоящее время по цене хорошего цветного телевизора можно приобрести в личное пользование достаточно мощную ЭВМ, за которую в 70-х годах потребовалось бы заплатить больше, чем за несколько самых дорогих автомобилей.

Несмотря на успехи, достигнутые в области технологии, существенных изменений в базовой структуре и принципах работы вычислительных машин не произошло. Так, еще в 1946 г. в описании впервые предложенной ЭВМ с хранимой в памяти программой отмечалось: «Мы располагаем... двумя личными типами памяти: памятью чисел и памятью команд. Тем не менее, и команды представлены машине в виде числового кода и, если машина каким-либо образом в состоянии отличать числа от команд, то блок памяти можно использовать для хранения и тех и других».

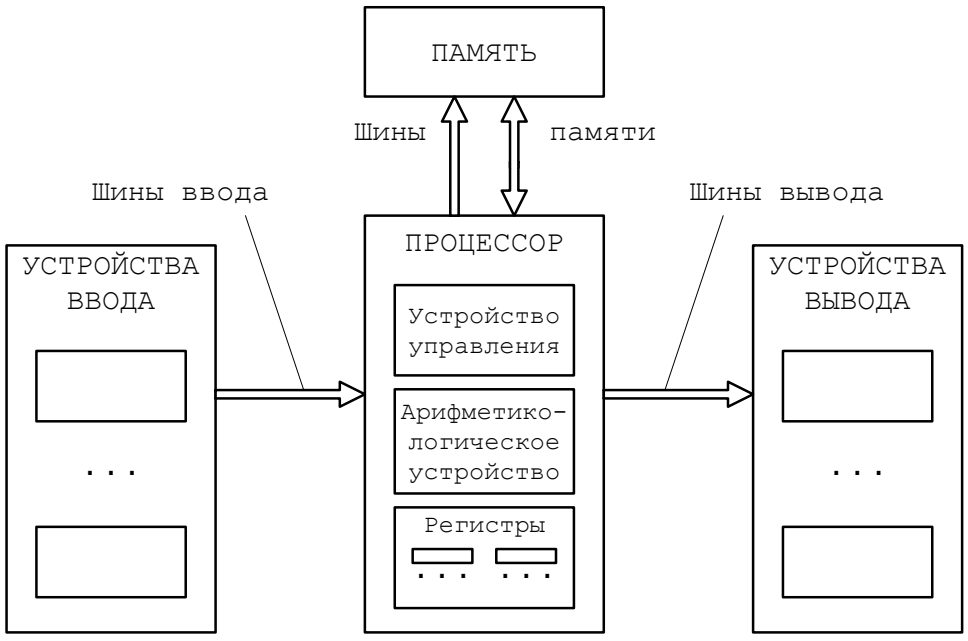


Рис. 1.6. Обобщенная структурная схема ЭВМ

Несмотря на успехи, достигнутые в области технологии, существенных изменений в базовой структуре и принципах работы вычислительных машин не произошло. Так, еще в 1946 г. в описании впервые предложенной ЭВМ с хранимой в памяти программой отмечалось: «Мы располагаем... двумя личными типами памяти: памятью чисел и памятью команд. Тем не менее, и команды представлены машине в виде числового кода и, если машина каким-либо образом в состоянии отличать числа от команд, то блок памяти можно использовать для хранения и тех и других».

И в настоящее время почти во всех вычислительных машинах для хранения данных (чисел, текстов) и команд служит одна и та же память. Это позволяет повысить эффективность использования достаточно дорогостоящей памяти ЭВМ, так как среди решаемых ею задач встречаются задачи с достаточно сложной обработкой (много команд) небольшого числа исходных данных (например, ряд научных расчетов) и задачи, связанные с переработкой по простым алгоритмам (мало команд) очень больших объемов данных (обработка переписи населения, расчет заработной платы на крупном предприятии и т. п.).

«Сердцем» ЭВМ является процессор, в состав которого входят:

- устройство управления выборкой команд из памяти и их выполнением;
- арифметико-логическое устройство, производящее операции над данными;
- регистры, осуществляющие временное хранение данных и состояний процессора;
- схемы для управления и связи с подсистемами памяти и ввода-вывода.

Подробнее о структуре процессора будет рассказано в конце этой главы.

Устройства ввода обеспечивают считывание информации (исходных данных и программы решения задачи) с определенных носителей информации (клавиатур, перфолент, магнитных лент или дисков, датчиков состояний управляемых объектов и т. п.) и ее представление в форме электрических сигналов, воспринимаемых другими устройствами ЭВМ (процессором или памятью).

Устройства вывода представляют результаты обработки информации в форме, удобной для визуального восприятия (индикаторы, печатающие устройства, графопостроители, экран дисплея и т. п.). При необходимости они обеспечивают запоминание результатов на носителях, с которых эти результаты могут быть снова введены в ЭВМ для дальнейшей обработки (перфоленты, магнитная лента, магнитный диск и т. п.), или передачу результатов

на исполнительные органы управляемого объекта (например, робота).

Память ЭВМ включает устройство, обеспечивающее хранение команд и данных. Это устройство состоит из блоков одинакового размера – ячеек памяти, предназначенных для хранения одного слова информации рис. 1.7, *а)*. В свою очередь, ячейка памяти состоит из элементов памяти, состояние каждого из которых соответствует одной двоичной цифре (0 или 1). Совокупность нулей и единиц, хранящихся в элементах одной ячейки, представляет собой содержимое этой ячейки памяти.

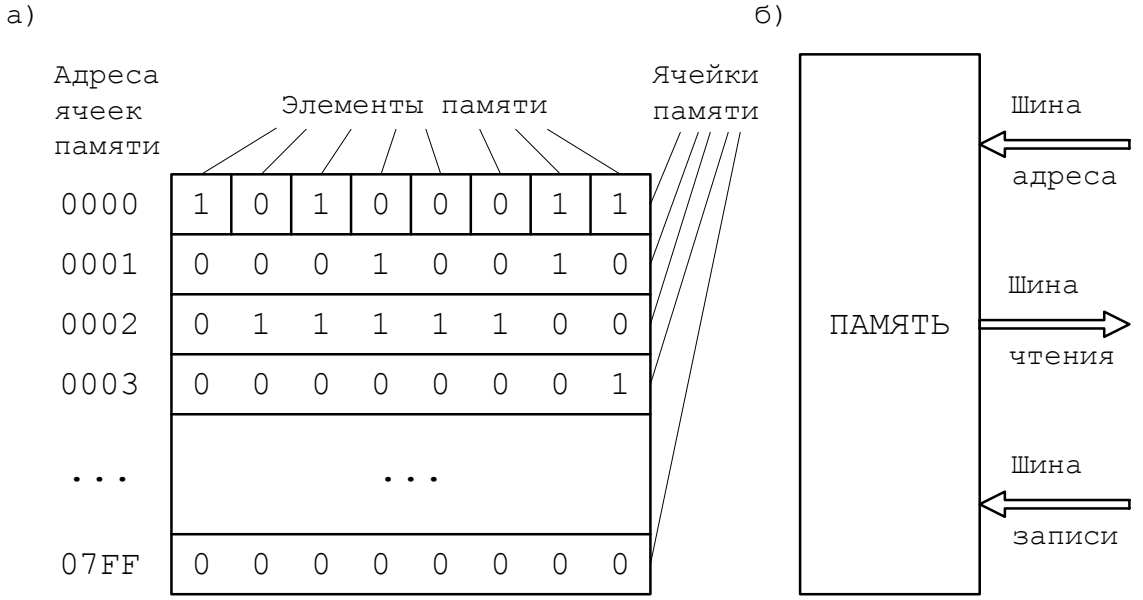


Рис. 1.7. Организация памяти ЭВМ

Ячейки нумеруются числами 0, 1, 2, называемыми адресом ячеек. Если необходимо записать в память слово, следует подать на шину адреса памяти (рис. 1.7,*б*) сигналы, соответствующие адресу ячейки, в которую надо поместить записываемое слово, и подать само слово на шину записи. Память устроена так, что заданное слово будет передано в ячейку с указанным адресом и может храниться там сколь угодно долго. В любой момент, обратившись к памяти, можно получить содержимое хранимого там слова. Для этого в память посылается адрес, определяющий местоположение требуемого слова, и она выдает по шине чтения копию слова. При считывании содержимое ячейки остается без изменения, так что, один раз записав слово, можно сколько угодно раз получать его копии. Это аналогично записи песни на магнитофонную ленту. Песню можно прослушивать с ленты (читать с ленты) сколько угодно раз, но если на ее место записать другую мелодию, то первая будет стерта. Однако время доступа к информации на магнитной ленте зависит от того, где записана эта информация (иногда надо перемотать почти

всю ленту, чтобы прослушать какую-либо песню), тогда как время доступа к любой ячейке памяти всегда одинаково (не зависит от ее номера).

На рис. 1.7,*а* показана память, имеющая $4096 = 2^{12}$ 8-разрядных слов, т. е. содержащая 4096 байт информации. При том же самом числе запоминающих элементов можно было бы организовать память из 32 768 1-битовых слов, 2048 16-битовых слов, 1024 32-битовых слов и т. д. Если нужно обрабатывать информацию, которая может кодироваться лишь двумя символами (например, некоторые данные переписи населения: пол, семейное положение и т. п.), то выгодно использовать первую организацию памяти. В случае же точных вычислений приемлемее память из 32-битовых или даже 64-битовых слов. Однако при выборе разрядности ячеек памяти надо учитывать, что в них должны храниться и команды программы, используемые процессором для обработки таких данных. А как же строится команда ЭВМ и какова ее разрядность?

Команда ЭВМ первоначально содержала следующую информацию (рис. 1.8,*а)*.

1. Код операции, указывающий операцию, которую должна выполнить ЭВМ (сложение, вычитание, умножение, сравнение, изменение знака и т. п.).

2. Адреса двух операндов – аргументов операции, например слагаемых, уменьшаемого и вычитаемого, сомножителей и т. п. Если какой-либо из операндов является константой, то вместо его адреса в команде может быть задано само значение операнда. Однако это обстоятельство должно быть отражено в коде операции, чтобы ЭВМ использовала соответствующую часть команды в качестве операнда, а не адреса ячейки памяти, в которой хранится этот операнд.

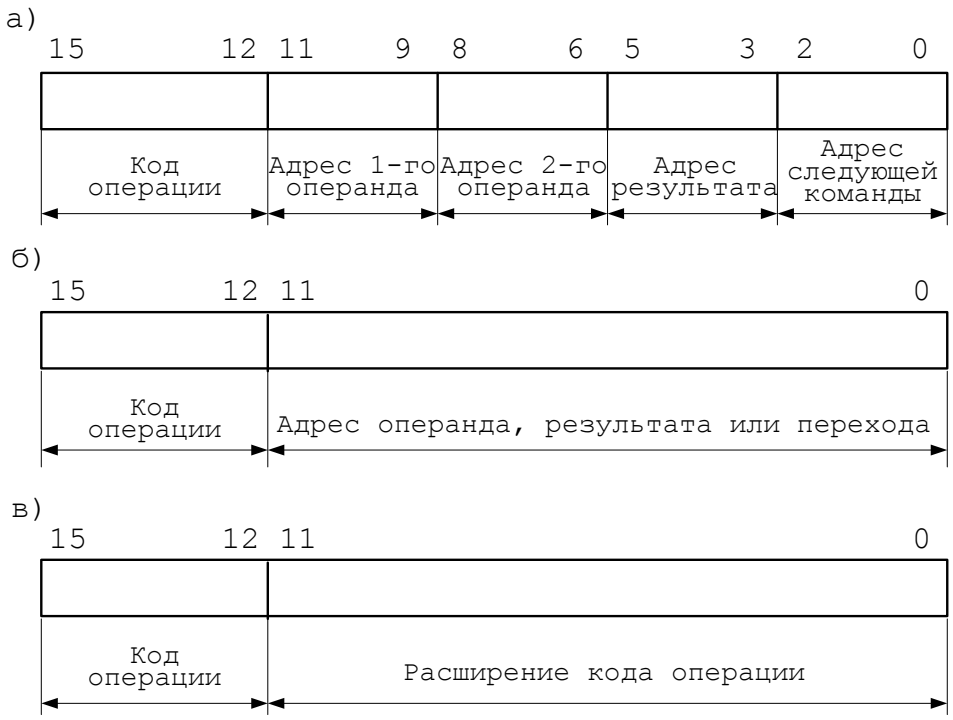


Рис. 1.8. Форматы команд:
а — четырехадресная; б — одноадресная; в — безадресная

3. Адрес ячейки памяти, в которую должен быть помещен результат операции.

4. Адрес следующей команды.

Такая команда, состоящая из пяти полей (код операции и четыре адреса), может быть реализована процессорами самых разнообразных структур.

Простейшая из этих структур (рис. 1.9) напоминает структуру процессора микрокалькулятора, приведенную на рис. 1.4. Процессор содержит устройство управления, АЛУ, регистр для размещения исполняемой команды (регистр команд) и регистр для размещения одного из операндов или результата операции в процессе выполнения этой команды – аккумулятор (на рис. 1.4, а роль аккумулятора выполнял регистр X). Рассмотрим, что происходит в процессоре после того, как в его регистр команд была переписана из памяти какая-либо команда, например команда вычитания.

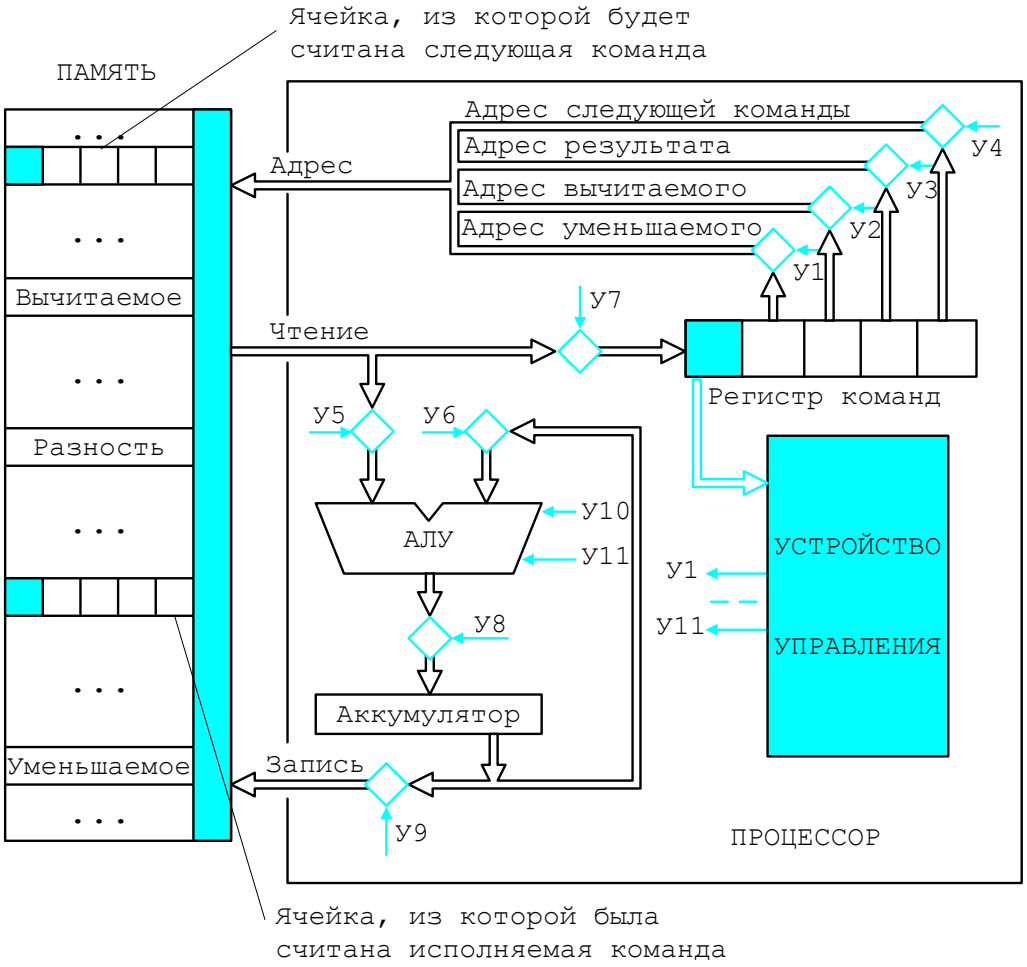


Рис. 1.9. Простой процессор, работающий с четырехадресной командой, целиком выбираемой из одной ячейки памяти

Так как поле регистра команд, в котором хранится код операции, связано шиной с устройством управления, то последнее получит приказ на выполнение операции «Вычитание» и перейдет в режим генерации управляющих сигналов:

- одновременной выработкой сигналов У1, У5, У10 и У8 обеспечиваются пересылка из памяти численного значения уменьшаемого, прохождение его через АЛУ (сигнал У10) и запись в аккумулятор;
- следующий тактовый импульс инициирует выработку сигналов У2, У5, У6 и У11, что приводит к пересылке из памяти численного значения вычитаемого, к выполнению операции вычитания этого значения (сигнал У11) из содержимого аккумулятора (из уменьшаемого);
- по сигналу У8 производится запись полученной разности в аккумулятор вместо уменьшаемого;

- затем сигналы У3 и У9 обеспечат пересылку разности из аккумулятора в ячейку памяти, на которую указывает предпоследнее поле команды;
- наконец, с помощью сигналов У4 и У7 будет произведена перепись в регистр команд следующей команды программы.

Рассмотренная структура процессора и реализуемая им структура четырехадресной команды кажутся вполне естественными. Действительно, без усложнения доступа к содержимому ячеек памяти нельзя одновременно получать из нее оба операнда и, следовательно, первый операнд приходится сохранять в процессоре до момента получения следующего операнда. Поэтому в состав процессора и включен аккумулятор (он назван так потому, что может быть использован для накопления – аккумуляции – слагаемых в процессе получения суммы многих слагаемых или накопления результатов других вычислений).

Нужен и регистр для хранения в процессоре исполняемой команды (регистр команд), так как во время выполнения этой команды из нее должна выбираться различная информация, используемая устройством управления и памятью. Однако целесообразно ли хранить в регистре команд одновременно все поля команды, если они все равно используются последовательно?

Когда команда хранится в одной ячейке памяти, то не существует возможности считывания из памяти отдельных полей этой команды. Поэтому лучше поставить вопрос так: целесообразно ли хранить четырехадресную команду в одной ячейке памяти или лучше хранить ее отдельные поля в нескольких ячейках памяти?

Короткую четырехадресную команду (см. рис. 1.8,а) безусловно следует целиком хранить в 16-битовой ячейке памяти. Но такая команда может адресовать лишь $2^3 = 8$ ячеек памяти. Современные же микроЭВМ адресуются к памяти, содержащей не менее $2^{16} = 65\ 536$ ячеек. Если создавать четырех-адресную команду, работающую с памятью подобных размеров, то на каждое адресное поле команды придется отвести 16 бит, а на всю команду (при 16-битовом коде операции) — 80 бит. Однако ЭВМ, работающую с такими командами, уже нельзя назвать микроЭВМ, так как она будет содержать очень много оборудования: 80-разрядные, ячейки памяти, 80 разрядные регистры, 80-разрядное АЛУ, 80-разрядные вентильные схемы, шины шириной 80 и т. д. Следовательно, надо либо четырехадресную команду хранить в ячейках памяти в виде отдельных ее полей, либо уменьшать число адресных полей команды.

Среди команд современных ЭВМ практически не встречаются четырехадресные. Мало и трехадресных команд (см. рис. 1.5,*г),* так как результат операции почти всегда можно записать на место одного из уже использованных операндов. Наибольшее

распространение в микроЭВМ получили одноадресные и безадресные команды (см. рис. 1.8,б,в), позволяющие построить простой процессор, например, показанный на рис. 1.10. Но прежде чем перейти к обсуждению таких коротких команд, ответим на ряд вопросов, которые, как нам кажется, могут возникнуть у любознательного читателя.

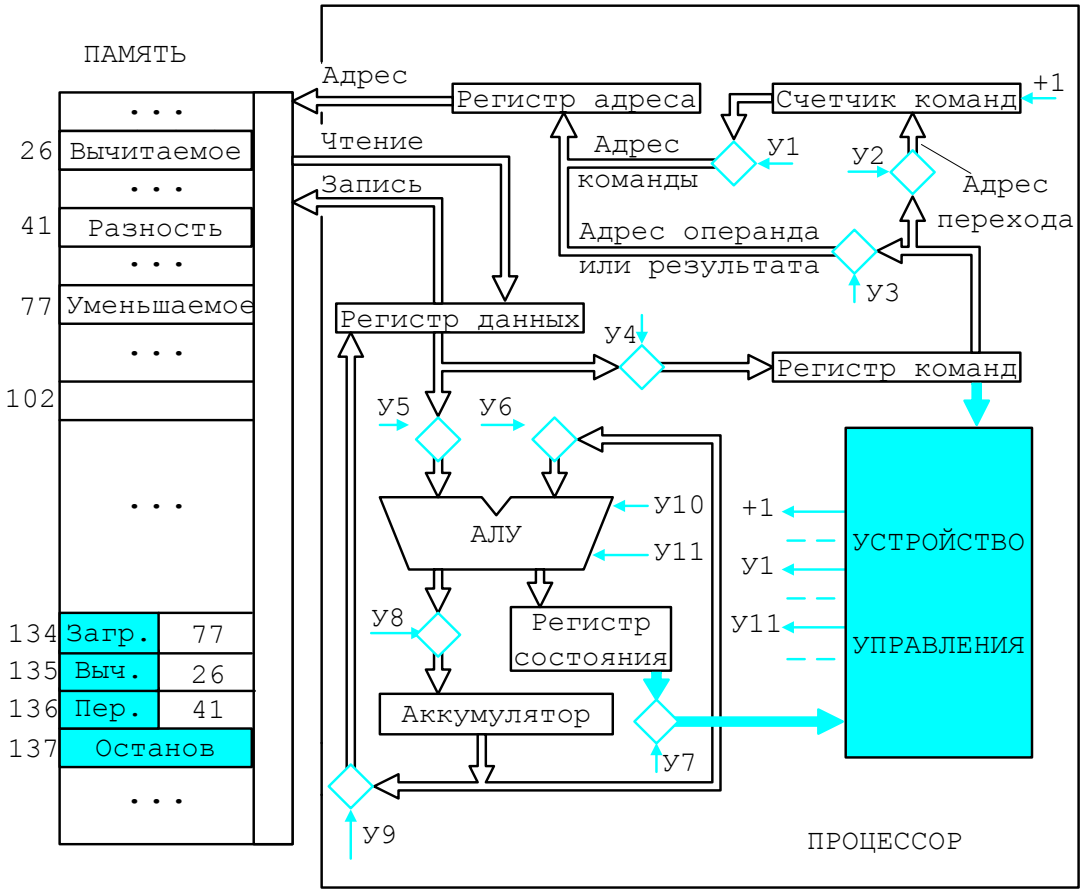


Рис. 1.10. Простой процессор, работающий с одноадресными и безадресными командами

Не потребуется ли хранить в разных ячейках памяти отдельные поля (или части полей) трех-, двух- или одноадресных команд и как работать с подобными командами?

Чем объясняется появление в процессоре, изображенном на рис. 1.10, дополнительных регистров (регистра адреса, счетчика команд и регистра данных)? Почему эти регистры связаны между собой так, как показано на рис. 1.10, а не иначе?

Если команды программы размещать в памяти ЭВМ друг за другом (а не в произвольной последовательности), то адрес следующей команды чаще всего будет отличаться от адреса исполняемой команды (или ее последнего поля) лишь на единицу, а добавление единицы к текущему адресу можно возложить на ЭВМ (счетчик команд на рис. 1.10). Это позволяет сократить длину команды (изъять из ее содержимого адрес следующей команды), но приводит к необходимости использования

специальных команд перехода, размещаемых в тех местах программы, где может потребоваться изменение естественной последовательности выполнения команд в зависимости от результата вычислений. Обычно команды перехода – одноадресные команды (см. рис. 1.8, б), где код операции оговаривает проверяемое условие (знак результата предыдущей операции, наличие переноса из старшего разряда и т. п.), а адрес – адрес команды, к которой нужно перейти, если условие выполняется (при невыполнении условия выбирается команда, расположенная вслед за командой перехода).

Теперь выясним, всегда ли нужны остальные адресные поля команды.

Анализ различных программ для ЭВМ показывает, что во многих случаях результат выполнения предыдущей команды используется как операнд в следующей. Если результат выполнения команды не пересылать в память, а сохранять, например, в аккумуляторе, то можно обойтись следующими одноадресными командами:

- командами арифметической (логической) обработки, которые дают приказ на выполнение какой-либо арифметической или логической операции, используя в качестве операндов содержимое аккумулятора и содержимое адресуемой ячейки памяти (точнее, содержимое регистра данных, куда переписывается содержимое адресуемой ячейки);
- командами пересылки, которые дают приказ на обмен информацией между аккумулятором и памятью (через регистр данных), т. е. на загрузку аккумулятора содержимым адресуемой ячейки памяти или запись в эту ячейку содержимого аккумулятора (результата вычислений);
- командами передачи управления, обеспечивающими условный или безусловный переход к адресуемой ячейке памяти (т. е. к .команде, которая должна быть помещена в процессор при выполнении какого-либо условия или вне зависимости от результата предыдущих вычислений), обращение к подпрограмме и выход из нее;
- командами ввода-вывода.

Как же с помощью таких команд будет выполняться, например, суммирование по формуле вида A = B + C + E + F + G + H? Для этого надо сначала загрузить в аккумулятор значение слагаемого B, затем выполнить пять одинаковых команд сложения и, наконец, команду пересылки содержимого аккумулятора в ячейку, выделенную для суммы A:

Команды программы		**Содержимое аккумулятора в процессе выполнения программы**
Загрузить	Адрес слагаемого B	B
Сложить	Адрес слагаемого C	B+C
Сложить	Адрес слагаемого E	B+C+E
Сложить	Адрес слагаемого F	B+C+E+F
Сложить	Адрес слагаемого G	B+C+E+F+G
Сложить	Адрес слагаемого H	B+C+E+F+G+H
Переслать	Адрес результата A	B+C+E+F+G+H
Останов		B+C+E+F+G+H

(Аналогичным образом Вы работаете с микрокалькулятором, когда, введя очередное слагаемое и нажав клавишу «+», добавляете это слагаемое к накопленной ранее сумме).

Отметим, что последняя из команд рассмотренной программы вообще не имеет адреса. Эта команда адресуется к устройству управления, а не к имеющей множество ячеек памяти. Подобные команды, предназначенные для работы с конкретными устройствами (аккумулятором, устройством управления и т. п.), называются безадресными командами или командами с неявной адресацией. Они позволяют сформировать приказы на обнуление аккумулятора, добавление к его содержимому константы (+1 или -1), перевод ЭВМ в состояние разрешения прерывания выполняемой программы и т. п.

Процессоры, реализующие программы из одноадресных и безадресных команд (см. рис. 1.8,б,в), являются простейшими и позволяют создавать наиболее дешевые ЭВМ. В одной из возможных структур такого процессора (см. рис. 1.10) используются уже знакомые нам устройства:

- арифметико-логическое устройство производит операции над двумя 16-разрядными величинами в целях получения 16-разрядного результата и выработки ряда признаков (результат меньше нуля, равен нулю или больше нуля, был перенос из старшего разряда результата);
- регистр состояния – 4-битовый регистр, в котором хранятся признаки результата последней операции, используемые командами перехода;
- аккумулятор – 16-разрядный регистр, в котором размещаются подлежащие обработке данные или результат обработки;
- регистр команд – 16-битовый регистр, служащий для размещения исполняемой команды;

- регистр адреса – 12-битовый регистр, содержащий адрес ячейки памяти, из которой будет считана команда (операнд) или записан результат обработки;
- регистр данных – 16-битовый регистр, используемый в качестве буфера между памятью и остальными регистрами процессора; через него пересылаются в процессор команды (операнды) и передаются в память результаты обработки;
- счетчик команд – 12-битовый регистр, содержимое которого увеличивается на единицу в момент выборки из памяти исполняемой команды и, если выбрана команда перехода, может быть заменено на содержимое адресной части команды перехода; в конце цикла исполнения команды в счетчике команд всегда хранится адрес той команды, которая должна исполняться вслед за текущей, а это может быть как следующая по порядку команда, так и команда, к которой требуется перейти при выполнении условий, заданных кодом операции команды перехода.

Для решения задачи на такой ЭВМ необходимо:

1) через устройства ввода информации загрузить в память ЭВМ программу решения задачи (алгоритм, написанный на языке ЭВМ) и исходные данные (это делается различными способами, о которых будет рассказано в следующих главах); и программа, и данные могут быть размещены в любой области памяти, начиная с ячейки 0 или другой ячейки с любым адресом (сначала программа, а затем данные, или наоборот).

2) «сообщить» процессору адрес ячейки памяти, в которой размещена первая команда программы, для чего занести адрес этой ячейки в счетчик команд (об одном из способов загрузки пускового адреса будет рассказано в параграфе 3.1);

3) нажать кнопку ПУСК, что приведет к передаче в память адреса первой команды программы и к пересылке ее содержимого из памяти в регистр команд; с этого момента процессор начнет выполнять последовательность достаточно простых операций, показанных на схеме алгоритма выполнения команды (рис. 1.11).

Как видно из схемы алгоритма, два первых действия (блоки 1 и 2) выполняются для каждой команды. Эти действия (а также действия по определению типа команды – блоки 3, 6 и 9) обычно называют «Выборкой команды». Последующие действия алгоритма («Исполнение команды») полностью зависят от того, какая это команда.

Рассмотрим исполнение команды, расположенной в ячейке 135 на рис. 1.10. Оно начинается в момент, когда в аккумуляторе

хранится значение уменьшаемого, записанное туда при выполнении предыдущей команды, а в счетчике команд — адрес 135.

1. Адрес, содержащийся в счетчике команд, переписывается в регистр адреса и далее в память ЭВМ (сигнал *У1),* а оттуда считывается команда *«Выч. 26»,* помещаемая в регистр команд (сигнал *У4)*.

2. Счетчик команд наращивается на единицу (сигнал +1), чтобы он указывал на команду (адрес 136), расположенную в памяти вслед за выбранной командой.

3. Устройство управления считывает из регистра команд код исполняемой команды, выясняет, что это команда вычитания, и переходит к ее выполнению.

4. Из регистра команд считывается (сигнал *У3)* и пересылается в память (через регистр адреса) адрес вычитаемого (26), а из памяти переписывается в регистр данных численное значение вычитаемого.

5. Выполняется операция вычитания содержимого регистра данных из содержимого аккумулятора (сигналы *У5, У6* и *У11),* а затем полученная разность переписывается с выхода АЛУ в аккумулятор (сигнал *У8),* заменяя значение уменьшаемого.

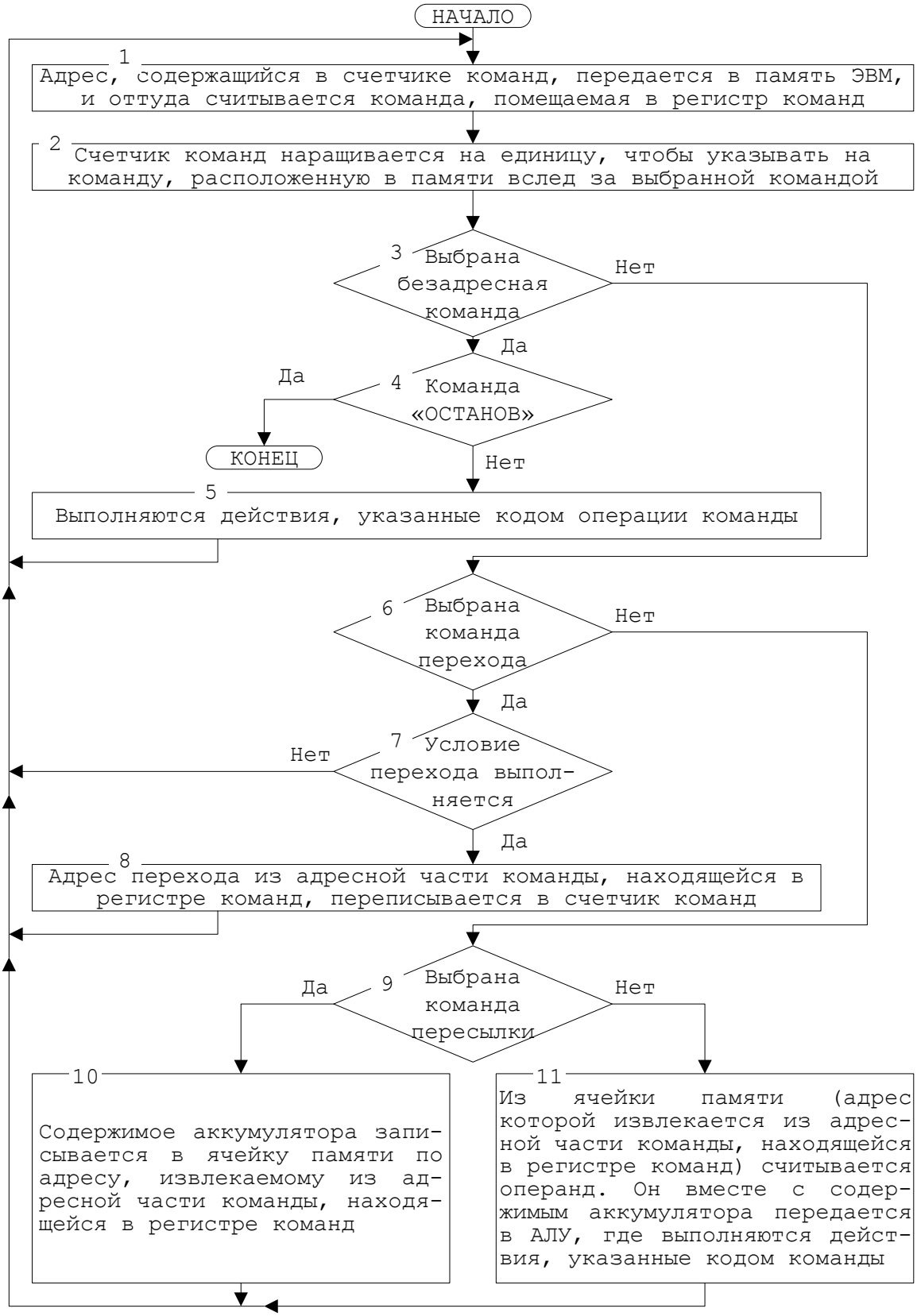


Рис. 1.11. Схема алгоритма выполнения команд простого процессора

Легко заметить, что процесс выполнения команды сводится к определению последовательности открывания и закрывания

вентильных схем. Описание того, какую вентильную схему и когда открывать, составляет программу для машины, система команд которой включает команду «Открыть вентильную схему». Такой машиной и является устройство управления ЭВМ, в котором хранятся программы реализации всех команд ЭВМ. Эти программы обычно называются микропрограммами, а их команды – микрокомандами. Простейшая микрокоманда состоит из набора битов, каждый из которых управляет одной из вентильных схем процессора: единица означает, что вентильная схема открыта, нуль – закрыта. С микропрограммным уровнем можно ознакомиться в гл. 4.

В заключение отметим, что процессор, изображенный на рис. 1.10, не является единственно возможным процессором, используемым для реализации одноадресных команд. Так, сумму произведений

$$A \cdot B + C \cdot E + F \cdot H + G \cdot K + \ldots$$

лучше вычислять с помощью процессора, имеющего два аккумулятора (один - для вычисления произведений, а другой - для накопления суммы). Это позволит исключить пересылку в память промежуточных результатов, т. е. ускорить вычисления и уменьшить число используемых ячеек памяти. Можно привести примеры, где целесообразно использовать процессор с большим числом аккумуляторов или других регистров (например, индексных). Процессоры, имеющие более двух аккумуляторов, называются процессорами с регистрами общего назначения (с РОНами).

Резюме

1. Структурно любая ЭВМ состоит из процессора, памяти и устройств ввода-вывода.
2. Память ЭВМ состоит из ячеек одинакового размера, каждая из которых имеет свой уникальный номер - адрес и может хранить команду или число. Количество элементов памяти (число битов), из которых составлена каждая ячейка, определяет разрядность машинных слов - команд и данных или их частей.
3. Все команды в качестве одной (может быть единственной) части содержат код операции, задающий действие, исполняемое по этой команде. Кроме кода операции команда может иметь несколько адресных частей, содержащих адреса операндов, результата и следующей команды.
4. В микроЭВМ используются безадресные, одноадресные и реже двухадресные команды. В одноадресных командах один из операндов выбирается из специального регистра - аккумулятора. В него же заносится и результат операции. Безадресные команды или задают какое-либо действие с устройствами ЭВМ (например,

останов), или используются для работы с операндами, имеющими фиксированное расположение (чаще всего с аккумулятором).

5. Процессор - это совокупность устройства управления, арифметико-логического устройства и связанных с ними регистров. Он организует процесс обработки информации путем выборки и последовательного выполнения команд программы, находящейся в памяти ЭВМ.

6. Арифметико-логическое устройство выполняет ряд арифметических и логических операций над одним или двумя операндами, пересылаемыми в него из памяти и (или) регистров процессора. Результат операции пересылается в один из регистров процессора.

7. Устройство управления, используя регистры процессора, управляет обменом информации между памятью, устройствами ввода-вывода и АЛУ, а также выполнением операций в АЛУ.

8. Регистры процессора служат для хранения промежуточных результатов вычислений и различной управляющей информации. Наиболее важные из них - это регистр команд (содержит исполняемую команду) и счетчик команд (содержит адрес следующей команды).

В процессе работы ЭВМ последовательно выполняет набор достаточно простых операций: выборку команды, определение ее типа, исполнение команды и определение адреса следующей команды. Разнообразие же решаемых на ЭВМ задач определяется не ее устройством, а программой и возможностями подключаемых к ЭВМ устройств ввода-вывода.

2. БАЗОВАЯ ЭВМ

2.1. Назначение и структура базовой ЭВМ

Вычислительные машины существенно различаются по возможностям, размерам и стоимости. Некоторые машины могут выполнять очень ограниченное число простых операций, в то время как другие – сотни различных операций (команд). Теоретически для решения конкретной задачи можно использовать любую ЭВМ. Малая будет решать задачу путем выполнения большого числа очень простых команд, в связи с чем может потребоваться большой, интервал времени для получения решения. Большая ЭВМ, способная выполнять множество операций, решит ту же задачу значительно быстрее.

В этой главе рассмотрена простая гипотетическая машина, обладающая типичными чертами многих конкретных микроЭВМ. Знание принципов построения и функционирования этой ЭВМ является хорошей базой для освоения микропроцессорных систем любых типов и моделей, поэтому она названа базовой ЭВМ. Естественно, начинать изучение ЭВМ целесообразно с подобной машины низкого уровня, что можно делать практически, используя ее модели, построенные для разных типов персональных ЭВМ. При построении базовой ЭВМ за прототип выбраны ЭВМ «Электроника 100» и «Саратов», а также схожие с ними ЭВМ типа PDP-8, однокристальный микропроцессор IM 6100 и персональная ЭВМ DECmate 11 [5, 6, 7].

На рис. 2.1 приведена упрощенная структура базовой ЭВМ. Это одноадресная машина, работающая с 16-разрядными словами. В ней реализованы два вида адресации операндов: прямая и косвенная. Рассмотрим составные части базовой ЭВМ, не касаясь пока устройств ввода-вывода (УВВ) и пульта управления (ПУ).

Память. Состоит память из 2048 ячеек по 16 бит с адресами 0, 1, $7FE_{16}$, $7FF_{16}$, Одну ячейку (или слово) можно использовать для хранения одного двоичного числа или одной команды программы, закодированной в двоичном коде. Пользователь должен сам определить, какая область памяти будет использоваться для данных и какая – для программы. При этом надо учитывать, что восемь ячеек памяти с адресами 008, 00F несколько отличаются от остальных. Эти ячейки называются индексными и они часто используются для организации циклов в программах.

Команды исходной программы обычно размещаются пользователем в соседних ячейках памяти. Машина считывает одну команду из памяти и выполняет указанную в ней операцию. Затем она считывает следующую команду из последующей ячейки и выполняет следующую операцию и т. д.

Процессор. В состав процессора входят:

- шесть регистров (16-разрядные регистр команд, аккумулятор и регистр данных, 11-разрядные счетчик команд и регистр адреса, 1-разрядный регистр переноса);

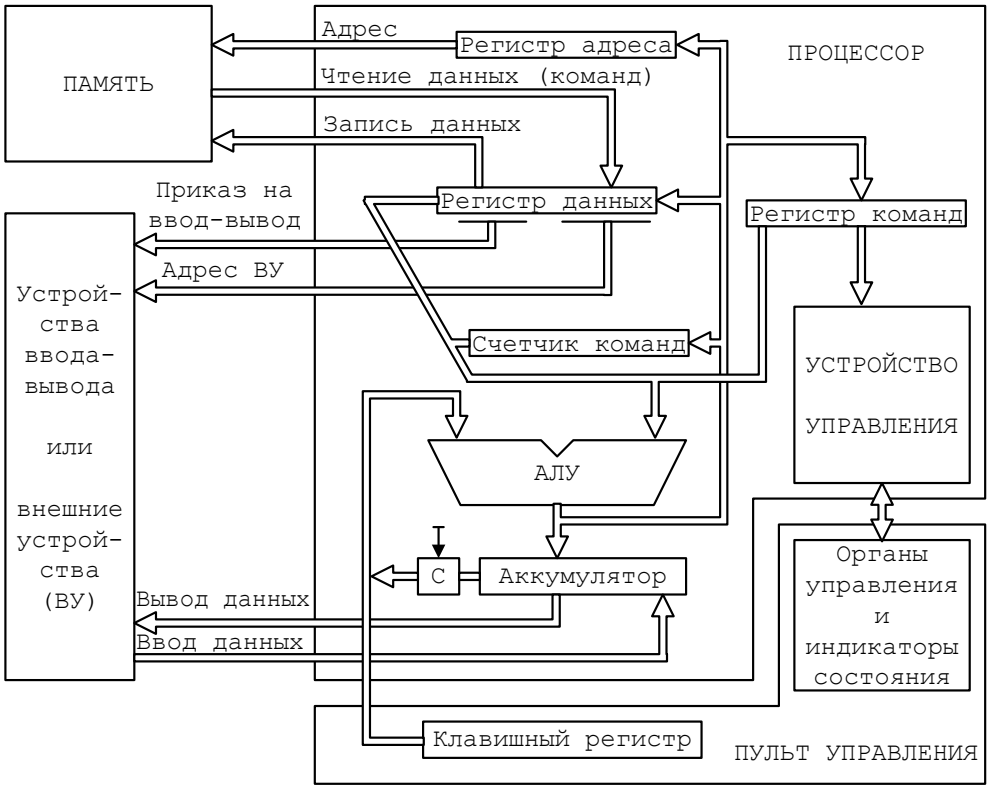


Рис. 2.1. Структура базовой ЭВМ

- арифметико-логическое устройство, выполняющее операции инвертирования любого операнда (или двух операндов), сложения двух 16-разрядных операндов (или их инвертированных значений), логического умножения двух 16-разрядных операндов, добавления к результату единицы и участвующее в организации циклических сдвигов содержимого аккумулятора и регистра переноса;
- микропрограммное устройство управления, хранящее в своей памяти микропрограммы выполнения 28 команд (см. табл. 2.4), четырех пультовых операций, реакции на сигналы прерывания выполняемой программы от устройств ввода-вывода и ряд других микропрограмм.

Рассмотрим подробнее назначение всех регистров процессора и АЛУ базовой ЭВМ.

Счетчик команд (СК). Он служит для организации обращения к ячейкам памяти, в которых хранится программа. В конце каждого цикла исполнения команды счетчик указывает адрес ячейки памяти, содержащей следующую команду программы. Так как команды чаще всего заносятся в последовательные ячейки, а при выполнении текущей команды содержимое счетчика увеличивается на единицу, то он

автоматически указывает адрес следующей команды. В некоторых случаях содержимое счетчика команд может быть изменено самой программой (например, при выполнении команд переходов). Таким образом, осуществляется передача управления другой части программы. В базовой ЭВМ используется 11-разрядный счетчик команд, который может формировать адреса любого из 2048 слов памяти.

Регистр команд (РК). Этот 16-разрядный регистр используется для хранения команды, непосредственно выполняемой машиной. Код операции команды (см., например, рис. 1.8, 1.10 или 2.3) пересылается из РК в устройство управления ЭВМ и декодируется. После этого происходят действия по реализации команды: считывание операнда и (или) выполнение операции, предписываемой командой,

Регистр адреса (РА). Он содержит значение исполнительного адреса ячейки памяти и состоит из 11 разрядов для адресации к 2048 ячейкам памяти. Если ЭВМ осуществляет выборку команды, то в РА пересылается содержимое счетчика команды СК для указания адреса ячейки, где хранится команда. Если ЭВМ производит выборку данных, то адрес может поступить из регистра команд.

Регистр данных (РД). Используется он для временного хранения 16-разрядных слов при обмене информацией между памятью и процессором. При считывании команды, числа или символа из памяти в процессор это слово сначала попадает в регистр данных, а потом пересылается либо в регистр команд (команды), либо в другие регистры процессора. При пересылке данных из процессора в память они сначала помещаются в РД и лишь затем записываются в нужную ячейку памяти.

Когда информационное слово находится в регистре данных, оно доступно для осуществления арифметических или логических операций. Содержимое РД может быть, например, сложено с содержимым аккумулятора, а полученный результат занесен в аккумулятор.

Аккумулятор (А). Регистр А является одним из главных элементов процессора. Машина может выполнять арифметическую или логическую операцию только над двумя операндами одновременно. Обычно первый операнд извлекается из памяти в регистр данных, в то время как второй находится в аккумуляторе. Операция, задаваемая командой, выполняется над содержимым РД и А, и результат операции помещается в аккумулятор.

Машина может проверить результат в аккумуляторе. В зависимости от результата проверки она может принимать различные решения. Здесь используется 16-разрядный аккумулятор. Всякий раз, когда при операции двоичного сложения возникает переполнение в старшем разряде, перенос теряется. Однако его можно записать в одноразрядный регистр переноса.

Регистр переноса (С). Это одноразрядный регистр, выступающий в качестве продолжения аккумулятора и заполняющийся при его

переполнении. Этот регистр также используется для организации циклических сдвигов. Состояние регистра переноса может проверяться для принятия решений.

Арифметико-логическое устройство (АЛУ). АЛУ базовой ЭВМ позволяет выполнять такие арифметические операции, как сложение и вычитание с учетом переноса (содержимого регистра С), полученного в результате осуществления предыдущей операции. Кроме того, оно способно выполнять операции логического умножения (операцию И), инвертирования, циклического сдвига и наращивания значения на 1

Если с помощью рассматриваемой ЭВМ надо, например, получить сумму чисел 53 и 106, то это можно сделать по программе, приведенной в табл. 2.1. Хотя и команды, и данные должны быть закодированы в двоичной форме (единственной форме, которую понимает ЭВМ), эта программа для простоты написана с символическим обозначением команд (подробнее о символическом кодировании см: в параграфе 2.2).

Используются следующие команды:

CLA (**Cl**ear **A**ccumulators – очистить аккумулятор) – по этой команде производится установка аккумулятора в нуль;

Таблица 2.1

Программа вычисления суммы двух чисел

		Комментарии
Адрес	Содержимое	
20	0053	Первое слагаемое
21	0106	Второе слагаемое
22	0000	Ячейка, предназначенная для результата
23	CLA	Аккумулятор содержит 0000 (очистка)
24	ADD 20	Аккумулятор содержит 53
25	ADD 21	Аккумулятор содержит 159 (53+106)
26	MOV 22	В ячейку с адресом 22 записывается 159
27	HLT	Останов ЭВМ (прекращение выборки команд)

ADD (**ADD** – сложить) – по этой команде содержимое ячейки с номером, написанным вслед за ADD, складывается с содержимым аккумулятора и результат остается в аккумуляторе;

MOV (**MOV**e – переслать) – по этой команде содержимое аккумулятора пересылается в ячейку с номером, написанным вслед за MOV; содержимое аккумулятора при этом сохраняется;

HLT (**H**a**LT** – стоп).

Программа записана в память ЭВМ, начиная с ячейки 23, а числовые данные – с ячейки 20. Ячейка с номером 22 отведена для записи суммы. Для выполнения программы необходимо установить в счетчик команд число 23 и пустить ЭВМ. Тогда выполнение программы начнется с чтения содержимого ячейки 23. Команда CLA установит аккумулятор в нуль. При выполнении этой команды содержимое счетчика команд увеличится на 1 и следующая команда будет считываться из ячейки 24. Это команда ADD 20

складывает содержимое ячейки 20 с содержимым аккумулятора, т. е. 53 с 0. По окончании выполнения команды в аккумуляторе содержится число 53, а в счетчике команд – 25 (он опять наращивается на 1).

Следующая команда читается из ячейки 25, выполняется суммирование содержимого ячейки 21 и аккумулятора, т. е. 106 и 53. По окончании команды в аккумуляторе содержится сумма 106 + 53= 159, а в счетчике команд – 26. Следующая команда – MOV 22 пересылает содержимое аккумулятора в ячейку с адресом 22. Теперь в этой ячейке и аккумуляторе содержится число 159, а в счетчике команд – 27. Из ячейки 27 выбирается команда HLT, счетчик команд наращивается на 1 (в нем содержится число 28), но по команде HLT прекращается выборка команд, т. е фиксируется окончание выполнения программы (содержимое ячейки 28 не выбирается в регистр команд и не интерпретируется как команда).

Рассмотрим теперь подробнее выполнение одной из команд этой программы, например команды ADD 21. Перед ее выполнением в регистре команд хранится предыдущая команда (ADD 20), в регистре адреса — адрес ее операнда (20), в регистре данных и аккумуляторе – значение этого операнда (53) и в счетчике команд – адрес рассматриваемой команды (25).

Сначала адрес команды пересылается из счетчика команд в регистр адреса (рис. 2.2, а). Здесь следует отметить, что в базовой ЭВМ все пересылки между регистрами выполняются через АЛУ (при необходимости пересылаемое значение инвертируется, к нему добавляется единица и т. п.). Затем из памяти в регистр данных выбирается содержимое ячейки памяти, адрес которой расположен в регистре адреса, и производится увеличение на единицу содержимого счетчика команд (рис. 2.2,б,в). Наконец, содержимое регистра данных пересылается в регистр команд, и устройство управления начинает анализировать это содержимое, т. е. команду ADD 21 (рис. 2.2,г). Перечисленные действия относятся к «Выборке команды».

Расшифровав код операции команды ADD 21, устройство управления пересылает ее адресную часть (адрес 21) в регистр адреса (рис. 2.2,д). Пересылка осуществляется из регистра данных, где еще сохраняется копия команды. Затем из памяти в регистр данных считывается содержимое ячейки памяти, адрес которой расположен в регистре адреса (рис. 2.2,е). Это действие аналогично действию на рис. 2.2,б, но устройство управления будет теперь рассматривать содержимое регистра данных не как команду, а как слагаемое, которое на следующем шаге складывается с содержимым аккумулятора (рис. 2.2,ж). Наконец, полученная в АЛУ сумма пересылается в аккумулятор (рис. 2.2,з) и тем самым завершается «Исполнение команды». Набор и порядок выполнения операций в этом цикле уникальны для каждой из команд ЭВМ и будут рассмотрены ниже.

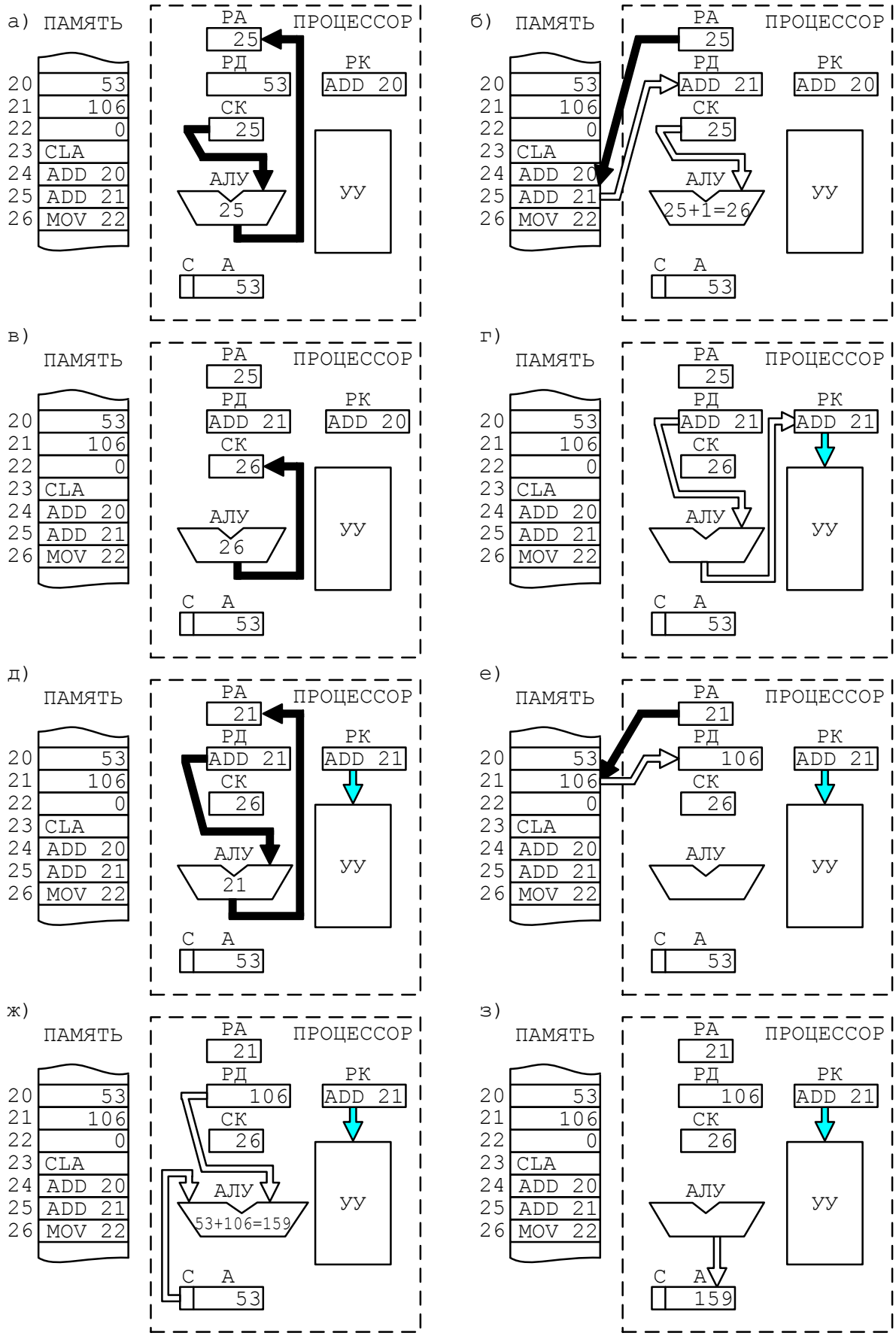


Рис. 2.2. Выполнение команды ADD 21

2.2. Кодирование программ и система команд

Все операции в ЭВМ выполняются над электрическими сигналами, закодированными двоичной цифрой 0 или 1. Поэтому перед занесением в память данные и команды должны быть вручную или аппаратно преобразованы в двоичную форму. Однако при написании программы команды могут быть записаны в любой другой форме (например, мнемонической, как в табл. 2.1). В общем случае программа может кодироваться четырьмя способами: в двоичной, восьмеричной, шестнадцатеричной и символической или мнемонической форме.

Двоичная форма команды. Эта форма является единственной, которую понимает машина. Каждой ЭВМ присуща система команд в двоичном коде, которая понятна ей. Регистр команд, куда помещается команда из памяти, связан электрическими цепями с устройством управления ЭВМ, где производится декодирование команды, и инициируются сигналы по реализации команды. В табл. 2.2 приведено содержимое части памяти ЭВМ, где хранятся программа и данные для суммирования двух чисел из примера параграфа 2.1. Левый столбец – это адреса ячеек в двоичной форме. Правый столбец представляет содержимое ячеек в двоичной форме. Адрес имеет 11 разрядов (память ЭВМ состоит из $2048 = 2^{11}$ ячеек), а каждая из ячеек – по 16 разрядов. Для удобства чтения двоичных чисел и их перевода в шестнадцатеричную систему они разбиты на тетрады.

Восьмеричное и шестнадцатеричное кодирование. Из предыдущего примера ясно, что записывать программу в двоичном коде очень неудобно и утомительно, если учесть, что средняя по сложности программа состоит из нескольких сотен или тысяч нулей и единиц. Для сокращения записи целесообразно использовать систему счисления с основанием $2^3 = 8$ или $2^4 = 16$ (см. параграф 1.3). Выбор для кодирования команд той или иной системы счисления в основном связан с форматом команд ЭВМ. В нашем случае удобнее шестнадцатеричная система, так как большинство полей в командах базовой ЭВМ (см. рис. 2.2) равно или кратно 4. В табл. 2.3 приведена та же программа, что и в табл. 2.1 и 2.2, только закодированная в шестнадцатеричной системе. Для программиста такое кодирование более удобно, чем двоичное.

Мнемоническое (символическое) кодирование. Шестнадцатеричное (или восьмеричное) кодирование имеет очевидное преимущество перед двоичным. Однако для длинных программ и оно неудобно. Программист должен выучить наизусть шестнадцатеричные коды всех команд, использующихся в машине (в

некоторых ЭВМ их более сотни). Чтобы упростить процесс написания, отладки и чтения программы, предложен мнемонический или символический код: каждая команда представляется простым двух-, трех- или четырехбуквенным мнемоническим символом. Мнемонические символы значительно легче связать с машинными операциями, так как их можно выбирать таким образом, чтобы они напоминали название команды. Большинство мнемонических кодов — это сокращения английских названий команд: SUB от **SUB**tract (вычесть), BR от **BR**anch (перейти), BPL от **B**ranch if **PL**us (перейти по положительному числу) и т. п. Намного легче запомнить, что инвертирование аккумулятора (**C**o**M**plement **A**ccumulator) кодируется CMA, чем запомнить двоичный код 1111010000000000 или даже его шестнадцатеричный эквивалент F400. Пример символического кодирования программы был приведен в параграфе 2.1 (см. табл. 2.1).

Таблица 2.2

Пример двоичного кодирования содержимого памяти

Ячейка	Содержимое
000 0010 0000	0000 0000 0101 0011
000 0010 0001	0000 0001 0000 0110
000 0010 0010	0000 0000 0000 0000
000 0010 0011	1111 0010 0000 0000
000 0010 0100	0100 0000 0010 0000
000 0010 0101	0100 0000 0010 0001
000 0010 0110	0011 0000 0010 0010
000 0010 0111	1111 0000 0000 0000

Таблица 2.3

Пример шестнадцатеричного кодирования содержимого памяти

Ячейка	Содержимое
020	0053
021	0106
022	0000
023	F200
024	4020
025	4021
026	3022
027	F000

Хотя символическое кодирование очень удобно для программиста, оно не может быть понято машиной. Единственным языком, понятным машине, является язык двоичных кодов. Следовательно, необходимо транслировать символическую программу в ее двоичный эквивалент. Это можно сделать вручную, используя таблицы соответствия (вида табл. 2.4). На практике трансляция осуществляется специальной машинной программой.

Классификация команд. ЭВМ способна понимать и выполнять определенный набор команд. При составлении программы программист ограничен этими командами. Количество и тип команд изменяются в зависимости от возможностей и назначения ЭВМ.

В зависимости от того, к каким блокам машины обращается команда или на какие блоки она ссылается, команды можно разделить на три группы: обращения к памяти (адресные команды); обращения к регистрам (регистровые или безадресные); ввода-вывода.

Команды обращения к памяти предписывают машине производить действия с содержимым ячейки памяти, адрес которой указан в адресной части команды. Например, команда ADD 20 из табл. 2.1 является командой обращения к памяти. Она предписывает машине обращение по адресу 20 и использование содержимого этой ячейки в качестве первого операнда. Второй операнд находится в аккумуляторе. Эти два операнда суммируются.

Таблица 2.4

Система команд базовой ЭВМ

Наименование	Мнемоника	Код	Описание
Адресные команды			
Логическое умножение	ADD M	1XXX	(M) & (A) ➔ A
Пересылка	MOV M	3XXX	(A) ➔ M
Сложение	ADD M	4XXX	(M) + (A) ➔ A
Сложение с переносом	ADC M	5XXX	(M) + (A) + (C) ➔ A
Вычитание	SUB M	6XXX	(A) - (M) ➔ A
Переход, если перенос	BCS M	8XXX	Если (C)=1, то M ➔ СК
Переход, если плюс	BPL M	9XXX	Если (A)>=0, то M ➔ СК
Переход, если минус	BMI M	AXXX	Если (A)<0, то M ➔ СК
Переход, если ноль	BEQ M	BXXX	Если (A)=0, то M ➔ СК
Безусловный переход	BR M	CXXX	M ➔ СК
Приращение и пропуск	ISZ M	0XXX	(M)+1 ➔ M; Если (M)>=0, то (СК)+1 ➔ СК
Обращение к п/программе	JSR M	2XXX	(СК) ➔ M; M+1 ➔ СК
Безадресные команды			
Очистка аккумулятора	CLA	F200	0 ➔ A
Очистка рег. переноса	CLC	F300	0 ➔ C
Инверсия аккумулятора	CMA	F400	($\overline{A}$) ➔ A
Инверсия рег. переноса	CMC	F500	($\overline{C}$) ➔ C
Циклический сдвиг влево на 1 разряд	ROL	F600	Содержимое A и C сдвигается влево, A(15) ➔ C и C ➔ A(0)
Циклический сдвиг вправо на 1 разряд	ROR	F700	Содержимое A и C сдвигается вправо, A(0) ➔ C и C ➔ A(15)
Инкремент аккумулятора	INC	F800	(A) + 1 ➔ A
Декремент аккумулятора	DEC	F900	(A) - 1 ➔ A
Останов	HLT	F000	
Нет операции	NOP	F100	
Разрешение прерывания	EI	FA00	
Запрещение прерывания	DI	FB00	
Команды ввода-вывода			
Очистка флага	CLF B	E0XX	0 ➔ флаг устройства B
Опрос флага	TSF B	E1XX	Если флаг устройства B равен 1, то (СК) + 1 ➔ СК
Ввод	IN B	E2XX	(B) ➔ A

Вывод	OUT B	E3XX	(A) ➔ B
Условные обозначения:			
(M),(A),(СК),(C) и (B) – содержимое ячейки с адресом M, аккумулятора, счетчика команд, регистра переноса и регистра данных устройства ввода-вывода с адресом B;			
A(X) – разряд аккумулятора с номером X;			
XXX – адрес ячейки памяти;			
XX – адрес устройства ввода-вывода.			

Безадресные команды выполняют различные действия без ссылок на ячейку памяти. Например, команда CLA из табл. 2.1 предписывает машине очистить аккумулятор. Эта команда имеет дело с операндом, расположенным в конкретном месте – в аккумуляторе. Другой пример безадресной команды — команда HLT из табл. 2.1.

Команды ввода-вывода осуществляют обмен данными между ЭВМ и внешними устройствами. В них задаются адрес (название) устройства ввода-вывода и код той операции, которую должно выполнить это устройство (приказ на ввод-вывод).

В табл. 2.4 дан перечень команд базовой ЭВМ. Подробно действия, выполняемые машиной по этим командам, рассмотрены в следующих параграфах настоящей главы.

Поясним одно из описаний: (СК) + 1 ➔ СК в командах ISZ и TSF. После выполнения этой операции счетчик команд будет указывать не на следующую команду программы, а на команду, расположенную за ней. Это произойдет потому, что после выборки команды ISZ или TSF содержимое счетчика команд уже было увеличено автоматически на единицу.

Существует и другой способ разделения команд на группы. Он основан на учете функций, выполняемых командой. Можно выделить восемь типов команд: пересылок или обмена, арифметические, логические, сдвигов, переходов, обращения к подпрограмме, управления и ввода-вывода

Форматы команд и способы адресации. В параграфе 1.4 рассматривались различные форматы (структуры) команд. Разработчики базовой ЭВМ выбрали три формата 16-битовых (однословных) команд с 4-битовым кодом операций (рис. 2.3).

С помощью 4-битового числа можно закодировать не более чем 2^4=16 различных операций. Разработчики отвели два кода (1110 и 1111) на команды ввода-вывода и безадресные команды. Нетрудно заметить, что в этих командах либо:

- используется меньшая по длине адресная часть (8-битовый адрес устройства ввода-вывода на рис. 2.3,*в)*,
- эта часть вообще отсутствует (рис. 2,3,*б*).

Следовательно, появилась возможность иметь до 2^4=16 команд ввода-вывода (4-битовый приказ на ввод-вывод) и до 2^{12} = 4096 безадресных команд (12-битовое расширение кода операции).

В командах обращения к памяти на адрес отведено 11 бит, что позволяет осуществить прямое адресование всех 2048 (2^{11}) ячеек памяти базовой ЭВМ.

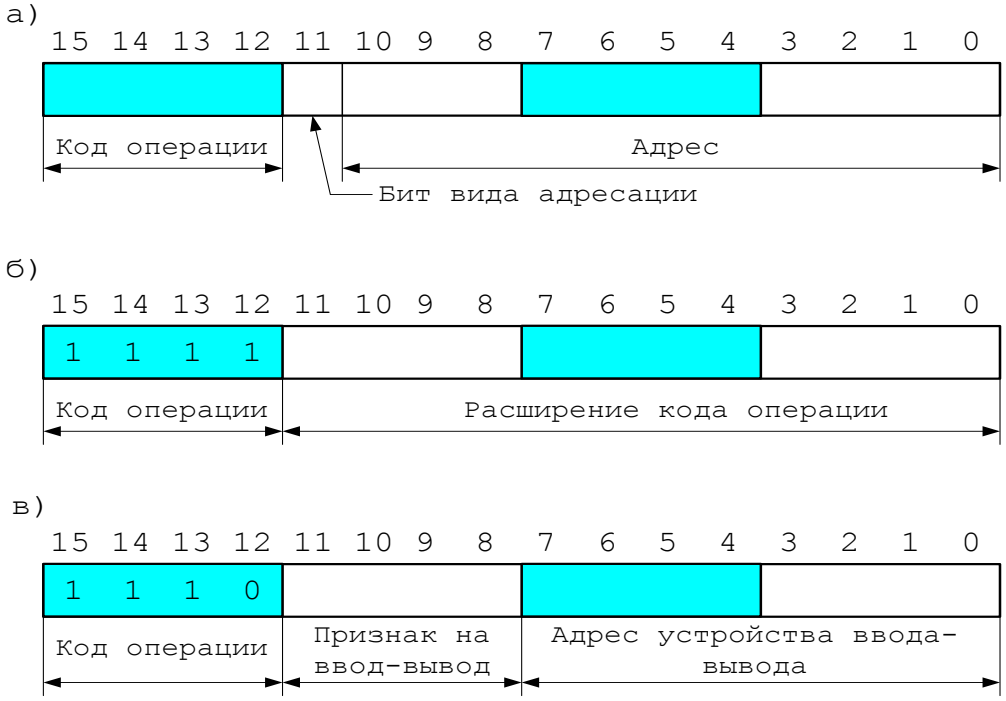


Рис. 2.3. Форматы команд базовой ЭВМ
а) адресных, б) безадресных, в) ввода-вывода

Однако встречаются приложения, когда в команде целесообразнее размещать не сам адрес операнда (результата или перехода), а его указатель, т. е. адрес ячейки памяти, в которой сохраняется адрес операнда (результата или перехода). Такое косвенное адресование упрощает построение циклических программ, организацию работы с подпрограммами, а также создает условия для расширения адресуемого пространства (косвенное адресование 16-битовых ячеек базовой ЭВМ позволяет ей иметь память объемом до 2^{16} = 65 536 слов).

Для указания вида адресации в командах используется бит с номером 11 (рис. 2.3,а), в который при прямой адресации следует записывать 0, а при косвенной – 1. В мнемонических изображениях команд для указания косвенной адресации операнд помещается в скобки. Так, на рис. 2.4 команда ADD 25 или 4025 указывает, что из ячейки 25 должно быть взято число 53, которое нужно сложить с содержимым аккумулятора. Команда же ADD (25) или 4825 указывает, что из ячейки 25 должен быть взят адрес ячейки 53, в

которой хранится число 47, которое и нужно сложить с содержимым аккумулятора. ("А это веселая птица-синица, которая ловко ворует пшеницу, которая в темном чулане хранится в доме, который построил Джек." Похоже?)

2.3. Арифметические операции

В этом параграфе рассматриваются основные способы записи чисел в базовой ЭВМ, арифметические операции, выполняемые с этими числами, и команды, реализующие такие операции. С рядом этих команд (CLA, ADD), командой пересылки (MOV) и командой останова (HLT) мы уже ознакомились.

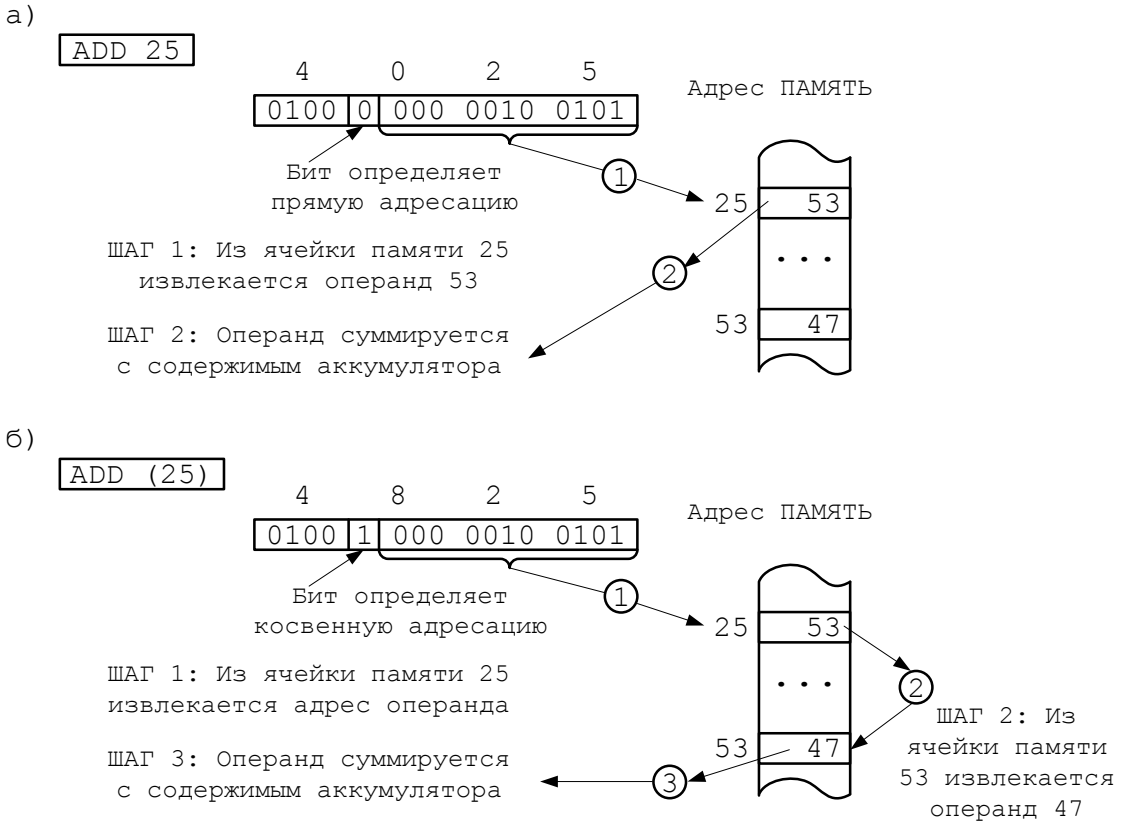


Рис 2.4. Способы адресации: *а* — прямой; *б* — косвенный

Целые двоичные числа без знака. Такие числа можно использовать для представления нуля и целых положительных чисел. При размещении этих чисел в одном 16-битовом слове (см. рис. 1.1) они могут изменяться от $(0000\ 0000\ 0000\ 0000)_2 = (0000)_{16} = 0$ до $(1111\ 1111\ 1111\ 1111)_2 = (FFFF)_{16} = 2^{16}-1 = 65535$. Подобные числа (так же, как и рассмотренные ниже целые двоичные числа со знаком) относятся к числам с фиксированной запятой, поскольку в них положение запятой, разделяющей целую и дробную части числа, строго фиксировано после младшего бита (см. рис. 1.1 и 1.5, а).

Суммирование целых двоичных чисел без знака выполняется по тем же правилам, что и десятичное сложение, за исключением того, что перенос в следующий разряд производится после того, как сумма достигает 2 (1 + 1), а не 10. Рассмотрим несколько примеров сложения.

Пример 2.1. Сумма меньше 65536 (2^{16}):

```
      1      111  1          11              Переносы
  + 0000 0110 0011 1010    + 1594
    0100 0100 1001 1011      17563
    0100 1010 1101 0101      19157
```

Пример 2.2. Сумма равна 65536:

```
  1                          111             Переносы
  + 1000 0000 0000 0000    + 32768
    1000 0000 0000 0000      32768
  1 0000 0000 0000 0000          0    ?
```

Здесь в ответе должно получиться число 32768 + 32768 = 65536, но поскольку разрядность слова составляет лишь 16 бит, то в нем сохраняется часть результата, т. е. 0. В данном и следующем примерах при сложении появляется единица переноса из старшего разряда, которая оказывается за пределами представления числа (в несуществующем 17 разряде). Для сохранения этой единицы в ЭВМ используется специальный 1-битовый регистр переноса - С (см. параграф 2.1). Пользователь ЭВМ должен так составлять программы, чтобы при выполнении следующего за суммированием действия нужным образом учитывалось (использовалось или игнорировалось) содержимое регистра переноса.

Пример 2.3. Сумма больше чем 65536:

```
  1 1111 1111 1111 111       111             Переносы
  + 1111 1111 1111 1111    +65535
    0111 1111 1111 1111      32767
  1 0111 1111 1111 1110      32766    ?
```

Здесь из-за ограниченной разрядности слова вместо 65535 + 32767 = 98302 опять (как и в примере 2.2) получен результат, уменьшенный на 65536 (2^{16}), и появляется перенос из старшего разряда.

В базовой ЭВМ для реализации рассмотренной операции используется команда ADD (СЛОЖЕНИЕ). Пример программы сложения двух чисел был рассмотрен в параграфах 2.1 и 2.2.

Целые двоичные положительные и отрицательные числа. Чтобы отличить отрицательные числа от положительных, в любых рассмотренных выше системах счисления можно использовать либо знак (такое представление чисел называется их прямым кодом), либо

величину числа (одно из таких представлений чисел называется их дополнительным кодом).

Использование чисел со знаком (прямого кода представления чисел) усложняет структуру ЭВМ, так как операция сложения двух чисел, имеющих разные знаки, должна быть заменена на операцию вычитания меньшей величины из большей и присвоения результату знака большей величины. Поэтому в большинстве ЭВМ отрицательные числа представляют в виде так называемых дополнений, что при суммировании двух чисел с разными знаками позволяет заменить вычитание на обычное сложение.

Дополнением М n-разрядного целого числа К называется разность

$$M = b^n - K, \qquad (2.1)$$

где b - основание системы счисления.

Так как количество различных целых n-разрядных чисел равно b^n (от 0 до b^n - 1), то половину этих чисел рассматривают как положительные (от 0 до $b^n/2$- 1), а другую половину - как отрицательные (т. е. как дополнения к первой половине чисел). Полный же набор таких чисел называют числами, представленными в дополнительном коде (табл. 2.5).

Легко проверить правильность получения дополнений (кодов отрицательных чисел) в столбцах табл. 2.5. Для этого надо сложить число из нижней половины таблицы с его дополнением, расположенным над кодом 0. Результатом такого суммирования будут числа $10^5=(1\ 00\ 000)_{10}$, $16^4 = (1\ 0000)_{16}$ или $2^{16}= (10\ 000\ 0000\ 0000\ 0000)_2$. В связи с тем, что разрядность чисел в табл. 2.5 ограничена 5, 4 и 16 разрядами соответственно, 1 в старшем разряде полученных сумм не учитывается, так как она "вываливается" за разрядную сетку. Следовательно, все суммы равны нулю, т. е. правильному результату сложения двух одинаковых по величине чисел, имеющих разные знаки.

Таблица 2.5

Дополнительные коды чисел

Прямой код ряда 5-разрядных десятичных чисел	Дополнительный код чисел		
	5-разрядных десятичных	4-разрядных шестнадцатеричных	16-разрядных двоичных
-50 000	50 000	–	–
-49 999	50 001	–	–
-49 998	50 002	–	–
…	…	…	…
-32 769	67 231	–	–
-32 768	67 232	8000	1 000 0000 0000 0000
-32 677	67 233	8001	1 000 0000 0000 0001
…	…	…	…
-3	99 997	FFFD	1 111 1111 1111 1101

-2	99 998	FFFE	1 111 1111 1111 1110
-1	99 999	FFFF	1 111 1111 1111 1111
0	00 000	0000	0 000 0000 0000 0000
+1	00 001	0001	0 000 0000 0000 0001
+2	00 002	0002	0 000 0000 0000 0010
+3	00 003	0003	0 000 0000 0000 0011
…	…	…	…
+32 766	32 766	7FFE	0 111 1111 1111 1110
+32 767	32 767	7FFF	0 111 1111 1111 1111
+32 768	32 768	–	–
…	…	…	…
+49 998	49 998	–	–
+49 999	49 999	–	–

Дополнения числа можно получить и без вычитания. Перепишем формулу (2.1) в виде

$$M = b^n - K = ((b^n - 1) - K) + 1. \quad (2.2)$$

Здесь число $b^n - 1$ состоит из n цифр (b - 1), т. е. старших цифр используемой системы счисления (для чисел в табл. 2.5 это 99 999, FFFF или 1111 1111 1111 1111). Поэтому $(b^n - 1) - K$ можно получить путем образования дополнений до b - 1 для каждой из цифр числа К в отдельности, а искомое дополнение - суммированием этого числа с 1. Так как в двоичной системе дополнения цифр до 1 соответствуют их инверсным значениям (дополнение 0 равно 1, а дополнение 1-0), то можно дать простое правило для получения дополнения двоичных чисел.

1. Получить инверсию заданного числа (все его 0 заменить на 1, а все 1 - на 0):

0 000 0010 1100 0101	число
1 111 1101 0011 1010	инверсия числа

2. Образовать дополнительный код заданного числа путем добавления 1 к инверсии этого числа:

1 111 1101 0011 1010	инверсия числа
+ 1	слагаемое 1
1 111 1101 0011 1011	дополнительный код числа

Проверим правильность перевода:

1 1 111 1111 1111 111	переносы
+ 0 000 0010 1100 0101	число
1 111 1101 0011 1011	дополнительный код числа
1 0 000 0000 0000 0000	0

Так как перенос из старшего разряда не учитывается, то результат суммирования равен 0, что подтверждает правильность преобразования.

В заключение отметим еще одну особенность дополнительного кода двоичных чисел: их старший бит выполняет функцию знака числа, т. е. равен 0 для положительных чисел и 1 – для их дополнений (отрицательных чисел). Поэтому старший разряд ячеек памяти и регистров, которые могут хранить такие числа, обычно называют знаковым разрядом.

Сложение целых двоичных чисел со знаком. Выполняется оно по тем же правилам и с помощью той же команды ADD, что и для чисел без знака. Примеры сложения целых чисел X и Y в целях получения суммы Z приведены в табл. 2.6.

В двух случаях (втором и четвертом) ограниченная разрядность чисел приводит к искажению не только величины, но и знака результата. Исправить положение можно, добавляя к результату (или вычитая из него) число $2^{16} = 65536$, но для этого надо переходить к размещению результата в двух словах.

Таблица 2.6

Различные случаи сложения двоичных чисел со знаком

Случай	Слагаемые и результат	Комментарии
1	$X>0, Y>0, X+Y < 2^{15}$ 0 000 0110 0011 1010 X=+ 1594 + 0 100 0100 1001 1011 Y=+17563 0 100 1010 1101 0101 Z=+19157	Результат корректный
2	$X>0, Y>0, X+Y \geq 2^{15}$ 0 100 0100 1001 1011 X=+17563 + 0 100 0010 1101 0101 Y=+19157 1 000 1111 0111 0000 Z=–28816	При сложении положитель-ных чисел получен отрицательный результат – ПЕРЕПОЛНЕНИЕ!
3	$X<0, Y<0, \lvert X\rvert+\lvert Y\rvert < 2^{15}$ 1 111 1001 1100 0110 X=– 1594 + 1 011 1011 0110 0101 Y=–17563 1 1 001 0101 0010 1011 Z=–19157	Результат корректный. Перенос из старшего раз-ряда не учитывается
4	$X<0, Y<0, \lvert X\rvert+\lvert Y\rvert > 2^{15}$ 1 011 1011 0110 0101 X=–17563 + 1 011 0101 0010 1011 Y=–19157 1 0 111 0000 1001 0000 Z=+28816	При сложении отрицатель-ных чисел получен положительный результат – ПЕРЕПОЛНЕНИЕ!
5	$X>0, Y<0, \lvert X\rvert>\lvert Y\rvert$ 0 100 0100 1001 1011 X=+17563 + 1 111 1001 1100 0110 Y=– 1594 0 011 1110 0110 0001 Z=+15969	Результат корректный. Перенос из старшего раз-ряда не учитывается
6	$X>0, Y<0, \lvert X\rvert<\lvert Y\rvert$ 0 000 0110 0011 1010 X=+ 1594 + 1 011 1011 0110 0101 Y=–17563 1 100 0001 1001 1111 Z=–15969	Результат корректный

Если слагаемые таковы, что вероятность получения искаженных результатов не равна нулю, то перед суммированием необходимо анализировать знаки слагаемых и результата. Разные знаки слагаемых или совпадение знаков слагаемых со знаком суммы свидетельствуют о том, что результат будет корректным. В противном случае вычисления должны быть прерваны и выдан сигнал ПЕРЕПОЛНЕНИЕ.

Размещение целого двоичного числа в нескольких словах памяти. Используется оно для расширения диапазона представления таких чисел и, следовательно, для увеличения точности расчетов. Так, размещая число в двух 16-битовых словах, мы сможем работать с числами в диапазоне

от $(1\ 000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_2 = -\ 2147483648 = 2^{31}$
до $(\underbrace{0\ 111\ 1111\ 1111\ 1111}_{\text{старшее слово}}\ \underbrace{1111\ 1111\ 1111\ 1111}_{\text{младшее слово}})_2 = \ 2147483647 = 2^{31}-1$

вместо диапазона от -32768 до +32767, в котором изменяются однословные двоичные числа со знаком (см. табл. 2.5).

Так как арифметико-логическое устройство базовой ЭВМ позволяет обрабатывать лишь 16-разрядные операнды, то для суммирования многословных чисел приходится выполнять определенную последовательность действий:

1) произвести сложение младших слов чисел, при этом в регистр переноса автоматически занесется 1 или 0 в зависимости от наличия или отсутствия переноса, возникающего при сложении старших разрядов суммируемых слов;

2) записать полученную сумму в младшее слово результата;

3) произвести сложение следующих по старшинству слов и содержимого регистра переноса, в конце операции в регистр переноса заносится перенос из старшего разряда суммируемых слов (1 или 0);

4) полученную сумму записать в соответствующее слово результата;

5) повторять пп. 3 и 4 до тех пор, пока не будут просуммированы все слова слагаемых.

В табл. 2.7 приведен пример программы сложения чисел X= 12345678 $(BC614E)_{16}$ и Y = 76543210 $(48FF4EA)_{16}$.

Таблица 2.7

Программа суммирования двухсловных чисел

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
0	F200	CLA	Очистить аккумулятор (аккумулятор содержит 0)
1	400A	ADD A	Аккумулятор содержит младшее слово слагаемого X
2	400C	ADD C	Аккумулятор содержит сумму младших слов X и Y, а регистр переноса – перенос, возникающий

3	300E	MOV E	при их сложении (в данном случае перенос равен 1)
4	F200	CLA	Сумма младших слов X и Y перемещается в младшее слово, отведенное для результата
5	5009	ADC 9	Очистить аккумулятор (подготовка к суммированию старших слов слагаемых)
6	400B	ADD B	Аккумулятор содержит сумму старшего слова X и содержимого регистра переноса, в котором хранится перенос, полученный при сложении
7	300D	MOV D	младших слов
8	F000	HLT	Аккумулятор содержит сумму старших слов X и Y, а также переноса
9	00BC		Сумма старших слов X и Y перемещается в
A	614E		старшее слово, отведенное для результата
B	048F		Останов ЭВМ (прекращение выборки команд)
C	F4EA		Старшее слово слагаемого X
D	0000		Младшее слово слагаемого X Старшее слово слагаемого Y
E	0000		Младшее слово слагаемого Y
			Ячейка, отведенная для старшего слова результата, сюда запишется число 1356 $(054C)_{16}$
			Ячейка, отведенная для младшего слова результата, сюда запишется число 22072 $(5638)_{16}$

В этой программе используется команда ADC B, обеспечивающая суммирование содержимого аккумулятора, ячейки с адресом B и регистра переноса, в котором хранится значение переноса (0 или 1), полученное при выполнении предыдущей операции. Результат суммирования сохраняется в аккумуляторе, а в регистр переноса заносится 1 или 0 в зависимости от наличия или отсутствия переноса из старшего разряда суммируемых слов.

Увеличение на единицу (INCrement) и уменьшение на единицу (DEC-rement). По команде INC к содержимому аккумулятора прибавляется единица, а по команде DEC – единица вычитается. Если при этом появляется перенос из старшего разряда, то в регистр переноса (C) заносится 1, в противном случае — 0.

Изменение знака числа. Выше уже говорилось, что для изменения знака числа, записанного в дополнительном коде, необходимо его инвертировать, а затем прибавить единицу к младшему разряду. На ЭВМ такую операцию можно выполнить с помощью команд CMA (инверсия содержимого аккумулятора) и INC.

В табл. 2.8 приведен пример программы для получения Y= -X.

Таблица 2.8

Программа изменения знака числа

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
0	F200	CLA	Очистить аккумулятор (аккумулятор содержит 0)
1	4006	ADD 6	Аккумулятор содержит X
2	F400	CMA	Аккумулятор содержит инверсию X
3	F800	INC	К содержимому аккумулятора добавляется 1 и в нем образуется дополнительный код числа X
4	3007	MOV 7	Число -X помещается в ячейку, отведенную для Y
5	F000	HLT	Останов ЭВМ (прекращение выборки команд)
6	6844		Число X
7	0000		Ячейка, отведенная для Y, после исполнения программы сюда будет записано –X = 97BC

Вычитание (X - Y). Оно может выполняться путем сложения уменьшаемого X и дополнительного кода вычитаемого Y. Однако это требует записи и исполнения нескольких команд (CLA, ADD Y, CMA, INC, ADD X). Для сокращения программы и времени выполнения вычитания в базовой ЭВМ предусмотрена команда вычитания SUB Y (CLA, ADD X, SUB Y), которая реализует те же действия за меньшее время.

При выполнении вычитания (так же, как и при выполнении сложения) возможно переполнение. Признаком возникновения переполнения является получение положительного результата при $X<0$ и $Y>0$ или отрицательного результата при $X>0$ и $Y<0$ (см. табл. 2.6).

Умножение и деление. В базовой ЭВМ нет команд для осуществления умножения и деления (АЛУ не выполняет этих операций). Поэтому пользователь должен организовывать получение произведения или частного программным путем.

При двоичном умножении частичное произведение сдвигается на один разряд влево для каждого последующего разряда множителя. Если множитель равен 0, то частичное произведение равно 0; если множитель равен 1, то частичное произведение равно множимому.

В качестве примера рассмотрим получение произведений 5x5 = 25, 5x10 = 50 и 5x3 = 15:

```
 101   (5)        101   (5)         101   (5)
  11   (3)       101   (5)        1010   (10)
 101               101              000
101                000              101
1111   (15)       101               000
                 11001   (25)      101
                                  110010   (50)
```

2.4. Управление вычислительным процессом, сдвиги и логические операции

Почти все программы ЭВМ должны обладать способностью проверять исходные данные, промежуточные и окончательные результаты вычислений, чтобы на основании результатов проверки изменять нужным образом последовательность выполнения операций. Например, следует должным образом отреагировать на ситуацию, когда при сложении положительных чисел возникает отрицательный результат (см. параграф 2.3).

Подобная задача решается в базовой ЭВМ при помощи команд перехода (BCS, BPL, BMI, BEQ, BR), команд ПРИРАЩЕНИЕ и ПРОПУСК (ISZ), ОСТАНОВ (HLT). Все эти команды (кроме HLT) являются адресными, т. е. в них указывается адрес той ячейки памяти, из которой должна быть выбрана следующая команда при выполнении того или иного условия. Если же условие не выполняется, то должна исполняться команда, расположенная вслед за данной командой управления. Как и в других адресных командах, здесь можно использовать косвенную адресацию, т. е. указывать адрес ячейки, из которой и будет выбираться нужный адрес перехода.

Разветвления в программах организуются с помощью команд перехода. Эти команды не изменяют состояния аккумулятора и регистра переноса. Они могут лишь изменить содержимое счетчика команд, поместив туда адрес, определяемый адресной частью команды.

BCS M (ПЕРЕХОД, ЕСЛИ ПЕРЕНОС). Переход к команде, расположенной в ячейке с адресом M, если содержимое регистра переноса равно 1.

BPL M (ПЕРЕХОД, ЕСЛИ ПЛЮС). Переход к команде, расположенной в ячейке с адресом M, если содержимое аккумулятора больше или равно нулю, т. е. в его старшем бите содержится 0.

BMI M (ПЕРЕХОД, ЕСЛИ МИНУС). Переход к команде, расположенной в ячейке с адресом M, если содержимое аккумулятора меньше нуля, т. е. в его старшем разряде содержится 1.

BEQ M (ПЕРЕХОД, ЕСЛИ НУЛЬ). Переход к команде, расположенной в ячейке с адресом M, если содержимое аккумулятора равно нулю т. е. во всех его разрядах содержится 0.

Если условие, указанное в перечисленных командах, не выполняется, то переход не может быть осуществлен и исполняется команда, расположенная за командой условного перехода.

BR M (БЕЗУСЛОВНЫЙ ПЕРЕХОД). Переход к команде, расположенной в ячейке с адресом M, осуществляемый при любых значениях аккумулятора, регистра переноса и других регистров ЭВМ.

Пример **2.4.** Получить модуль числа X, содержащегося в ячейке памяти с адресом 100. Результат поместить в ячейку с адресом 200. Программа получения модуля приведена в табл. 2.9.

Таблица 2.9

Программа получения модуля

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
0 1 2	F200 4006 9005	CLA ADD 100 BPL 5	Очистить аккумулятор (аккумулятор содержит 0) Аккумулятор содержит аргумент X Если X ≥ 0, то перейти к команде, расположенной в ячейке с адресом 5
3 4	F400 F800	CMA INC	Изменение знака числа X, представленного в дополнительном коде
5 6	3200 F000	MOV 200 HLT	Запись |X| в ячейку с адресом 200 Останов ЭВМ (прекращение выборки команд)

Циклические программы используются в тех случаях, когда требуется несколько раз выполнить набор одинаковых действий над некоторым изменяющимся составом данных.

Базовая ЭВМ обладает рядом средств для упрощения организации циклических программ. Целесообразность введения этих средств удобнее всего рассмотреть на примерах.

Сначала рассмотрим получение произведения Z=50Y. Так как в системе команд базовой ЭВМ нет команды умножения, то воспользуемся простейшим способом: будем 50 раз складывать значение Y. При этом линейная программа, содержащая 50 команд ADD Y, вряд ли займет первое место в конкурсе программ для умножения. Ее, наверное, опередит программа, приведенная в табл. 2.10.

Существуют ли пути упрощения этой программы? Да.

Например, вместо подсчета и проверки количества выполненных циклов можно подсчитывать и проверять количество циклов, оставшихся до завершения работы программы, т. е. уменьшать в цикле значение M до тех пор, пока оно не станет равным нулю. Тогда отпадает необходимость в использовании ячейки 8 и команды вычитания (SUB 7) – команды с 15 по 19 заменяются последовательностью команд: ADD 7, DEC, MOV 7, BEQ 1A, BR 10. Если же в ячейку 7 поместить отрицательное значение множителя –50 = = (FFCE) 16, то команды с 15 по 19 можно заменить последовательностью команд: ADD 7, INC, MOV 7, BMI 10, HLT, т. е. освободить еще и ячейку 1A.

Таблица 2.10

Первый вариант программы для получения Z=50Y

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
5	0078	Y	Множимое (здесь десятичное значение 120)
6	0000	Z	Ячейка, отведенная для накопления результата. В ней поочередно будут храниться значения Y, 2Y, 3Y,… После 50 суммирований в ней будет содержаться искомый результат 50Y
7	0032	M	Множитель $50=(32)_{16}$
8	0000	C	Ячейка, используемая для накопления числа выполняемых циклов – счетчик циклов
10	F200	CLA	К промежуточному результату, находящемуся в ячейке 6, добавляется ещё одно значение множимого Y
11	4006	ADD 6	
12	4005	ADD 5	
13	3006	MOV 6	
14	F200	CLA	Содержимое счетчика циклов увеличивается на 1, а его копия пока сохраняется в аккумуляторе
15	4008	ADD 8	
16	F800	INC	
17	3008	MOV 8	
18	6007	SUB 7	Если содержимое счетчика циклов меньше значения множителя, то выполняется переход к командам, осуществляющим очередное суммирование Y с промежуточ-ным значением Z. В противном случае выполняется следующая команда (команда HLT)
19	A010	BMI 10	
1A	F000	HLT	Останов ЭВМ. В ячейке 6 содержится результат

Большего упрощения программы можно добиться при накоплении результата непосредственно в аккумуляторе, а не в ячейке 6. Тогда команды с 10 по 13 можно было бы заменить всего одной командой ADD 5. Однако в обсуждаемом варианте программы (см. табл. 2.10) аккумулятор пришлось использовать еще и для изменения числа циклов, что привело к необходимости сохранения промежуточного результата Z в памяти ЭВМ. Следовательно, для упрощения циклических программ целесообразно иметь в составе команд базовой ЭВМ команду, позволяющую организовать счетчик циклов без затрагивания аккумулятора. Такой командой является ISZ.

Команда ISZ (ПРИРАЩЕНИЕ И ПРОПУСК) служит для увеличения на 1 содержимого адресуемой ячейки памяти и перехода к одному из двух путей продолжения программы в зависимости от знака этого содержимого. Так, при каждом выполнении команды ISZ M, расположенной по адресу A, к содержимому ячейки с адресом M добавляется 1, и если результат меньше нуля, то выполняется команда, следующая за ISZ M (команда, расположенная по адресу A+1); в противном случае эта команда пропускается, т. е. выполняется

команда, расположенная по адресу A+2. Очень ценным свойством команды ISZ M является то, что она не портит содержимого аккумулятора и регистра переноса (увеличение M производится без использования этих регистров).

В табл. 2.11 приведен вариант программы получения Z = 50Y, в котором использованы все обсужденные выше возможности сокращения памяти и времени для накопления результата (время уменьшается за счет того, что в каждом из 50 циклов выполняются лишь три команды вместо десяти).

Таблица 2.11

Второй вариант программы для получения Z=50Y

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
5	0078	Y	Множимое
6	0000	Z	Ячейка, отведенная для результата
7	FFCE	M	Отрицательное значение множителя (-50)
10	F200	CLA	Очистка аккумулятора
11	4005	ADD 5	К содержимому аккумулятора добавляется Y
12	0007	ISZ 7	Содержимое M наращивается на 1, и если оно ещё
13	C011	BR 11	меньше 0, то выполняется команда BR 11. При M=0 ко-манда BR 11 пропускается
14	3006	MOV 6	Результат 50 сложений Y записывается в ячейку 6
15	F000	HLT	Останов ЭВМ

В качестве следующего примера рассмотрим получение в ячейке 005 суммы 32 элементов массива, размещенного в ячейках памяти с 010 по 02F.

В отличие от предыдущей задачи, где многократно суммировалось содержимое одной ячейки (Y), здесь надо суммировать содержимое разных ячеек. Естественно, нас не может устроить программа, содержащая последовательность из 32 команд сложения: 4010, 4011, 402E, 402F. Очевидно, что требуется организовать циклический процесс, в котором при каждом прохождении цикла должен изменяться адрес суммируемого элемента массива (первый раз этот адрес должен быть равен 10, при втором прохождении цикла — 11, при третьем — 12 и т. д.). Как же изменять адрес суммируемого элемента?

В первых ЭВМ изменение адреса осуществлялось путем модификации адресной части команды (переадресации): после того как содержимое какой-либо ячейки памяти было использовано ЭВМ в качестве команды, последующие команды программы использовали это содержимое в качестве операнда (числа) и преобразовывали его нужным образом (например, наращивали на 1). Воспользуемся этим приемом для решения поставленной задачи (табл. 2.12).

Таблица 2.12

Первый вариант программы суммирования элементов массива

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
5 6	0000 FFE0		Ячейка, отведенная для накопления результата. Отрицательное число элементов массива (-32)
10 … 2F	…	…	Численные значения элементов массива
30 31 32 33 34 35 36 37 38 39 3A	F200 4005 4010 3005 F200 4032 F800 3032 0006 C030 F000	CLA ADD 5 ADD 10* MOV 5 CLA ADD 32 INC MOV 32 ISZ 6 BR 30 HLT	Промежуточный результат (ячейка 5) суммируется с со-держимым элемента массива, адрес которого расположен в адресной части команды, находящейся в ячейке 32 (сначала этот адрес равен 10, а затем он при каждом прохождении цикла увеличивается на 1 с помощью команд 34-37) Пересылка в аккумулятор команды, расположенной по адресу 32, добавление к её содержимому 1 и запись мо-дифицированной команды на старое место (в ячейку 32) Наращивание на 1 счетчика элементов массива и переход к команде 30, пока его содержимое меньше нуля Останов ЭВМ

* Команда модифицируется во время выполнения программы

Переадресация команд практически не используется в современных ЭВМ. Для сближения языка команд с алгоритмическими языками были разработаны специальные средства адресации (часть из них кратко рассматривалась в параграфе 2.2). Эти же средства позволяют создавать эффективные программы, помещаемые в постоянные запоминающие устройства, откуда можно лишь читать команды или данные, но нельзя их изменять (программы микроЭВМ, управляющих работой стиральных машин, некоторых устройств автомобиля, роботом и т. п.). Одно из таких средств — косвенная адресация (см. рис. 2.4).

При использовании косвенной адресации нужно выбрать в памяти ЭВМ какую-либо ячейку (например, 007), записать в нее адрес первого элемента суммируемого массива (адрес 010), заменить в программе табл. 2.12 команду 4010 на команду 4807 (ячейка 32) и команды переадресации (CLA, ADD 32, INC, MOV 32) командами вычисления текущего адреса суммируемого элемента массива (CLA, ADD 7, INC, MOV 7). Скобки в команде ADD (7) и цифра 8 в ее числовом коде 4807 (единица в 11-м бите команды) информируют ЭВМ о том, что содержимое аккумулятора должно суммироваться не с содержимым

ячейки 7, а с содержимым ячейки, адрес которой хранится в ячейке 7. Поэтому в ячейку 7 сначала помещается адрес первого элемента суммируемого массива, а затем (при каждом прохождении цикла) этот адрес увеличивается на 1.

В табл. 2.13 приведен укороченный вариант программы суммирования элементов массива, где кроме косвенной адресации используется и новое средство для изменения текущего адреса суммируемого элемента массива. Этим средством является команда ISZ 7, при выполнении которой содержимое ячейки 7 наращивается на 1. Но так как это содержимое (адрес элемента массива) — положительная величина, то после выполнения ISZ 7 будет пропущена следующая за ней команда. Поэтому в ячейку 33 помещена команда NOP (НЕТ ОПЕРАЦИИ), но можно было бы поместить любое число.

Наконец, рассмотрим еще одно средство, позволяющее упростить циклические программы базовой ЭВМ, — индексные ячейки (ячейки с адресами от 008 до 00F). Если произвести косвенное адресование какой-либо из этих ячеек, то сначала ее содержимое будет использовано в качестве адреса операнда, а затем оно автоматически увеличится на единицу. При прямом адресовании индексные ячейки работают как обычные ячейки (их содержимое может измениться лишь в случае записи информации в ячейку).

Таблица 2.13

Второй вариант программы суммирования элементов массива

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
5 6 7	0000 FFE0 0010		Ячейка, отведенная для накопления результата. Отрицательное число элементов массива (-32) Текущий адрес элемента массива (сначала адрес первого элемента массива – 10)
10 … 2F	… 	… 	Численные значения элементов массива
30 31 32 33 34 35 36 37	F200 4807 0007 F100 0006 C031 3005 F000	CLA ADD (7) ISZ 7 NOP ISZ 6 BR 31 MOV 5 HLT	Очистка аккумулятора Суммирование очередного элемента массива Текущий адрес элемента массива наращивается на 1 Команда НЕТ ОПЕРАЦИИ Наращивание на 1 счетчика элементов массива и переход к команде 31, пока его содержимое меньше нуля Запись результата в ячейку 5 Останов ЭВМ

Указанное свойство индексных ячеек позволяет составить оптимальную программу для суммирования элементов массива (табл. 2.14).

Побитовая обработка данных обеспечивается в базовой ЭВМ командами логического умножения (AND), циклического сдвига влево (ROL) и вправо (ROR), а также командами инвертирования (CMC) и очистки (CLC) регистра переноса. Эти команды часто используются для упаковки информации в памяти ЭВМ (например, размещения в одном слове нескольких небольших чисел или кодов) и ее распаковки, обработки текстовой информации, умножения (деления) на 2, 4, 8 ... и т. п.

Команда AND M выполняет над каждым разрядом содержимого аккумулятора и содержимым ячейки памяти с адресом M булеву операцию «И». Результат выполнения команды для каждой пары битов операндов равен единице только тогда, когда оба бита равны единице; в остальных случаях бит результата равен нулю. Так как аналогичный результат получается и при поразрядном умножении битов (0x0 = 0, 0x1 = 0, 1x0 = 0 и 1x1 = 1), то команда называется логическим умножением. Эти свойства позволяют выделять или очищать определенные биты слова.

Таблица 2.14

Третий вариант программы суммирования элементов массива

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
5	0000		Ячейка, отведенная для накопления результата.
6	FFE0		Отрицательное число элементов массива (-32)
…	…	…	
F	0010		Текущий адрес элемента массива
10 … 2F	…	…	Численные значения элементов массива
30	F200	CLA	Очистка аккумулятора
31	480F	ADD (F)	Суммирование очередного элемента массива. Так как сначала в индексную ячейку F помещен адрес первого элемента массива (10), то после первого выполнения дан-ной команды содержимое ячейки F увеличится на 1 и будет указывать на второй элемент массива, после второго выполнения – на третий элемент и т.д.
32	0006	ISZ 6	Наращивание на 1 счетчика элементов массива и переход к команде 31, пока его содержимое меньше нуля
33	C031	BR 31	
34	3005	MOV 5	Запись результата в ячейку 5
35	F000	HLT	Останов ЭВМ

В качестве последнего примера упрощения циклических программ рассмотрим получение количества нечетных элементов массива, расположенного в ячейках с адресами от 010 до 02F. Программа для решения этой задачи приведена в табл. 2.15.

Таблица 2.15

Программа определения количества нечетных элементов массива

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
5	0000		Ячейка, отведенная для накопления результата.
6	0001		Маска для выявления нечетных элементов массива. При её логическом умножении на нечетное число результат будет равен 1 (иначе – 0)
7	FFE0		Отрицательное число элементов массива (-32)
8	0010		Текущий адрес элемента массива
10 … 2F	…	…	Численные значения элементов массива
30	F200	CLA	Очистка аккумулятора
31	4808	ADD (8)	Аккумулятор содержит очередной элемент массива
32	1006	AND 6	
33	4005	ADD 5	Выделяется младший бит аккумулятора (элемента масс-сива) и добавляется к сумме числа нечетных элементов массива. Если младший бит равен 1 (нечетный элемент), то сумма увеличивается на эту единицу
34	3005	MOV 5	
35	0007	ISZ 7	
36	C030	BR 30	
37	F000	HLT	Модификация содержимого счетчика элементов массива и переход по адресу 30, пока это содержимое меньше 0 Останов ЭВМ

Команды ROL (ЦИКЛИЧЕСКИЙ СДВИГ ВЛЕВО НА 1 РАЗРЯД) и ROR (ЦИКЛИЧЕСКИЙ СДВИГ ВПРАВО НА 1 РАЗРЯД) замыкают аккумулятор и регистр переноса в кольцо и сдвигают всё биты кольца на один разряд влево или вправо (рис. 2.5).

Рис. 2.5. Циклические сдвиги: а — влево; б — право

Сдвигами битов числа влево или вправо можно реализовать операции умножения или деления на 2 (один сдвиг), 4 (два сдвига) и т. д. Действительно, после сдвига числа

$$(0000\ 0000\ 0011\ 1010)_2 = 58$$

на один разряд влево будет получено число

$$(0000\ 0000\ 0111\ 0100)_2 = 116,$$

а после еще одного такого сдвига - число

$$(0000\ 0000\ 1110\ 1000)_2 = 232$$

(здесь предполагалось, что регистр переноса был очищен перед сдвигами).

Рассмотрим еще один пример упрощения программы за счет использования команд сдвигов: проанализировать элементы массива X (ячейки 010–02F) и при отрицательном X(1) присвоить H(1)-му элементу массива H значение 1, а при положительном X(1) – значение 0 (табл. 2.16).

Таблица 2.16

Программа выделения знака числа

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
7	FFE0		Отрицательное число элементов массивов X и H
8	0010		Текущий адрес элемента массива X
9	0030		Текущий адрес элемента массива H
10 … 2F	…	…	Численные значения элементов массива X
30 … 4F	…	…	Численные значения элементов массива H
50	F200	CLA	Очистка аккумулятора
51	4808	ADD (8)	В аккумулятор очередной элемент массива X
52	F600	ROL	Сдвиг знакового разряда X(I) в регистр переноса
53	F200	CLA	Очистка аккумулятора
54	F600	ROL	Сдвиг содержимого регистра переноса в младший разряд аккумулятора и запись полученного значения в H(I)
55	3809	MOV (9)	
56	0007	ISZ 7	
57	C050	BR 50	Модификация содержимого счетчика элементов массива и переход по адресу 50, пока это содержимое меньше 0
58	F000	HLT	Останов ЭВМ

В некоторых приложениях возникает необходимость инвертирования или очистки определенного бита слова. Для этого, например, можно:

- поместить слово в аккумулятор;
- сдвинуть слово так, чтобы нужный бит попал в регистр переноса;
- выполнить команду CMC ("Инвертирование регистра переноса") или CLC ("Очистка регистра переноса");
- сдвинуть биты слова на старое место.

2.5. Подпрограммы и параметры

Достаточно часто встречается ситуация, когда отдельные части программы должны выполнять одни и те же действия по обработке данных (например, вычисление тригонометрической функции). В подобных случаях повторяющиеся части программы выделяют в подпрограмму, а в соответствующие места программы заносят лишь команды обращения к этой подпрограмме. В базовой ЭВМ для этой цели используется команда JSR ("Обращение к подпрограмме").

На рис. 2.6 показана часть основной программы, содержащая две команды JSR 300, с помощью которых осуществляется переход к выполнению команд подпрограммы.

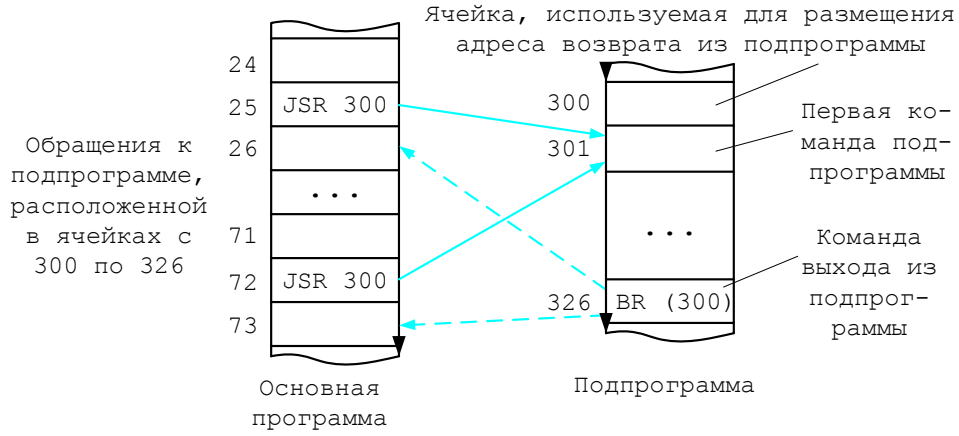


Рис. 2.6. Обращение к подпрограмме из основной программы

По команде JSR 300, расположенной в ячейке 25, выполняются запись числа 25+1=26 в ячейку с адресом 300 и переход к первой исполняемой команде подпрограммы, расположенной в ячейке 301. Далее начинается процесс выполнения команд подпрограммы, который завершается на команде BR (300), расположенной в ячейке 326. Это команда безусловного перехода с косвенной адресацией. Она предписывает ЭВМ выполнить переход к команде, расположенной по адресу, сохраняемому в ячейке 300. Так как в эту ячейку ранее было занесено число 26, то будет исполняться команда, записанная в ячейке 26, т. е. следующая за обращением к подпрограмме.

По команде JSR 300, расположенной в ячейке 72, осуществляется запись числа 72+1=73 в ячейку с адресом 300 и начинается выполнение подпрограммы (с ячейки 301). При исполнении последней команды подпрограммы BR (300) будет осуществлен переход к команде, расположенной по адресу 73, т. е. адресу, хранимому в ячейке 300.

Таким образом, при оформлении подпрограммы перед первой ее командой следует разместить ячейку, в которую будет пересылаться адрес возврата из подпрограммы. В обращении к подпрограмме указывается адрес именно этой ячейки, и команда JSR M выполняет следующие действия:

1) помещает в ячейку M адрес следующей за JSR M команды, т. е. команды, которая должна выполняться вслед за командами подпрограммы;

2) передает управление команде, расположенной в ячейке с адресом M+1.

Последней исполняемой командой подпрограммы должна быть команда выхода, т. е. команда BR (M), по которой осуществляется переход к команде с адресом, хранимым в ячейке M.

Пример 2.5. Получить удвоенный модуль чисел X, Y и Z, хранящихся в ячейках памяти 58, 63 и 71 соответственно. Результаты (2|X|, |Y|, 2|Z|) поместить в ячейки 74, 77 и 82.

Для решения этой задачи можно взять за основу программу примера 2.4, оформив ее фрагмент в виде подпрограммы (табл. 2.17). В программе табл. 2.17 в подпрограмму передается всего одна величина: в аккумулятор помещается число, модуль которого нужно удвоить. Результат вычислений (опять одна величина) возвращается в основную программу через тот же аккумулятор. А как же передать и (или) получить два или большее число параметров?

Таблица 2.17

Первый вариант программы получения удвоенного модуля трех чисел

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
Основная программа			
10 11 12 13	F200 4058 2030 3074	CLA ADD 58 JSR 30 MOV 74	Запись в аккумулятор значения X, обращение к подпрог-рамме получения удвоенного модуля (запись в ячейку 30 адреса 13 и переход к команде, расположенной по адресу 31), пересылка результата в ячейку для хранения 2 |X|
14 15 16 17	F200 4063 2030 3077	CLA ADD 63 JSR 30 MOV 77	Те же действия, что и в командах 10–13, но над аргументом Y. При обращении к подпрограмме в ячейку 30 пересылается адрес возврата 17
18 19 1A 1B	F200 4071 2030 3082	CLA ADD 71 JSR 30 MOV 82	Те же действия, что и в командах 10–13, но над аргументом Z. При обращении к подпрограмме в ячейку 30 пересылается адрес возврата 1B
1C	F000	HLT	Останов ЭВМ
Подпрограмма получения удвоенного модуля			
30 31	0000 9034	 BPL 34	Ячейка для размещения адреса возврата Первая команда: проверяет знак числа в аккумуляторе, и если он положительный, то передает управление командам удвоения содержимого аккумулятора
32 33	F400 F800	CMA INC	Изменение знака
34 35	F600 C830	ROL BR (30)	Удвоение содержимого аккумулятора Выход из подпрограммы (переход к выполнению команды, адрес которой хранится в ячейке 30)

В микроЭВМ, имеющих несколько аккумуляторов или, лучше сказать, регистров общего назначения, можно передавать в подпрограмму (или получать из нее) большее число параметров. Но и число регистров общего назначения также ограничено. Поэтому во всех микроЭВМ (как и в базовой ЭВМ) есть возможность обмена параметрами через ячейки памяти.

В табл. 2.18 приведен второй вариант программы для решения примера 2.7.

Здесь использован один из возможных способов передачи параметров: размещение параметров непосредственно за командой JSR. Адрес возврата, записываемый командой JSR, в действительности является адресом первого параметра. В подпрограмме приходится производить наращивание этого адреса для выборки остальных параметров и получения действительного адреса возврата - адреса команды, расположенной за последним из параметров. В качестве параметров можно использовать константы или переменные (адреса). В последнем случае несколько усложняется обработка этих параметров (см. табл. 2.18).

Таблица 2.18

Второй вариант программы получения удвоенного модуля трех чисел

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
Основная программа			
10 11 12	2030 0058 0074	JSR 30	Обращение к подпрограмме, за которым располагаются данные: адрес X и ячейки для размещения 2 |X|. В ячейку 30 передается адрес 11
13 14 15	2030 0063 0077	JSR 30	Обращение к подпрограмме, за которым располагаются данные: адрес Y и ячейки для размещения 2 |Y|. В ячейку 30 передается адрес 14
16 17 18	2030 0058 0074	JSR 30	Обращение к подпрограмме, за которым располагаются данные: адрес Z и ячейки для размещения 2 |Z|. В ячейку 30 передается адрес 17
19	F000	HLT	Останов ЭВМ
Подпрограмма получения удвоенного модуля			
30	0000		Ячейка для размещения адресов аргумента, результата и возврата из подпрограммы
31 32 33 34 35	F200 4830 3035 0030 0000	CLA ADD (30) MOV 35 ISZ 30	В ячейку 35 пересылается значение переменной, на кото-рую указывает содержимое ячейки 30 (при обращении к подпрограмме это содержимое является адресом аргумента), а затем адрес в ячейке 30 наращивается на 1 (теперь он указывает на адрес результата)
36	F200	CLA	В ячейку 3A пересылается адрес результата, а

37 38 39 3A	4830 3035 0030 0000	ADD (30) MOV 3A ISZ 30	адрес в ячейке 30 еще раз увеличивается на 1 (теперь он указывает на адрес той команды основной программы, которая расположена вслед за обращением к данной подпрограмме)
3B 3C 3D 3E 3F 40 41	F200 4835 9040 F400 F800 F600 383A	CLA ADD (35) BPL 40 CMA INC ROL MOV (3A)	Получение удвоенного модуля числа, адрес которого указывается содержимым ячейки 35, и пересылка результата по адресу, на который указывает содержимое ячейки 3A
42	C830	BR (30)	Выход из подпрограммы

Возможно и размещение параметров в специальной области параметров с передачей в подпрограмму начального адреса этой области. Основная программа может использовать эту область для нескольких подпрограмм, что позволит сократить объем памяти.

2.6. Выполнение машинных команд

Время, затрачиваемое на получение работоспособной программы ЭВМ, резко сокращается, если программист хорошо представляет, как выполняются отдельные машинные команды, и использует эти знания при отладке программы. Подробные сведения о процессах, происходящих в базовой ЭВМ во время выполнения отдельных команд, приведены в параграфах 4.2- 4.4, а здесь кратко рассмотрим лишь те из них, знание которых позволяет быстрее локализовать ошибки при исполнении отлаживаемой программы.

В процессе исполнения машинных команд устройство управления ЭВМ производит анализ и пересылку команды, отдельных ее частей (кода операции, признака адресации и адреса) или операнда из одного регистра машины в другой ее регистр, АЛУ, память или устройство ввода-вывода.

Эти действия (микрооперации) протекают в определенной временной последовательности и скоординированы между собой. Для создания временной последовательности (см. рис. 1.4) используется генератор тактовых импульсов (импульсов с частотой в несколько сотен или тысяч МГц).

Цикл команды. Для реализации одной команды требуется выполнить определенное количество микрокоманд, каждая из которых инициирует одновременное осуществление одной или нескольких микроопераций за время одного рабочего такта ЭВМ. Общее число тактовых импульсов (микрокоманд), требуемых для выполнения команды, определяет время ее выполнения, называемое циклом команды (рис. 2.7). Цикл команды обычно включает один или несколько машинных циклов.

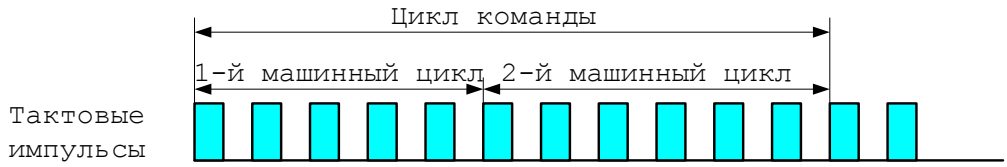


Рис. 2.7. Типичный цикл команды

Устройство управления базовой ЭВМ может находиться в четырех возможных состояниях: выборки команды, выборки адреса, исполнения и прерывания. Длительность каждого из этих четырех состояний определяет время выполнения соответствующего машинного цикла. Каждый машинный цикл предназначен для определенной цели.

Выборка команды. В данном машинном цикле выполняются чтение команды из памяти, ее частичное декодирование и иногда исполнение (для безадресных команд и команд ввода-вывода, являющихся одноцикловыми командами):

1) содержимое ячейки памяти, указываемой счетчиком команд, читается из памяти в регистр данных (см. рис. 2.2,*а,б*);

2) содержимое счетчика команд увеличивается на 1 (см. рис. 2.2,*б,в*);

3) содержимое регистра данных пересылается в регистр команд (см. рис. 2.2,*г*), код операции команды частично декодируется для выяснения типа команды (адресная, безадресная или ввода-вывода), анализируется бит признака адресации и происходит подготовка цепей, необходимых для выполнения команды;

4) если выбрана адресная команда, то осуществляется переход к микрокомандам следующего машинного цикла; в противном случае выполняются действия по завершению одноцикловой команды.

На рис. 2.8,*а,б* показаны действия, выполняемые в конце цикла выборки для одноцикловой команды СМА, инвертирующей содержимое аккумулятора (5).

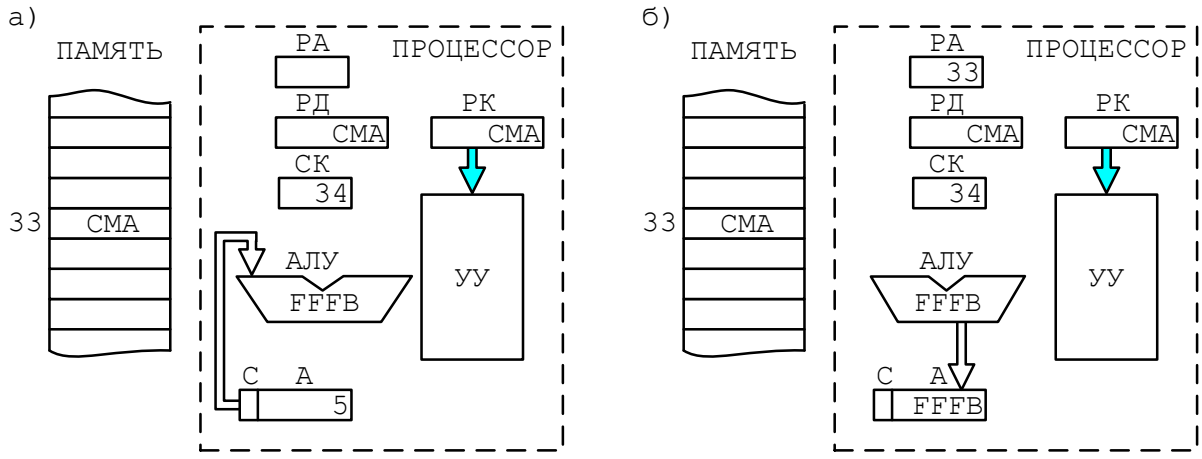


Рис. 2.8. Исполнение команды СМА в конце цикла выборки команды

Выборка адреса. Это состояние (этот машинный цикл) следует за выборкой команды для адресных команд с косвенной адресацией (бит вида адресации равен единице). Состояние используется для чтения адреса операнда (адреса результата или перехода) из памяти и состоит из следующих шагов:

1) адресная часть команды пересылается из регистра данных (где пока еще сохраняется копия команды) в регистр адреса (рис. 2.9,*а*);

2) содержимое ячейки памяти, указываемой регистром адреса, читается в регистр данных (рис. 2.9,*б*); теперь в этом регистре находится адрес операнда (адрес результата или перехода), который будет использоваться при выполнении команды (в цикле исполнения).

Для иллюстрации действий, осуществляемых в этом машинном цикле, выбрана команда ADD (21), расположенная в ячейке 43. После выборки команды в регистре данных и регистре команд хранится сама команда (4821): четыре старших разряда - код операции $(0100)_2$, затем признак адресации $(1)_2$ и, наконец, адрес $(000\ 0010\ 0001)_2 = (21)_{16}$. В счетчике команд записан адрес следующей команды (44), в аккумуляторе - результат предыдущих операций (7) и в 11-разрядном регистре адреса - адрес исполняемой команды (43). В начале цикла выборки адреса содержимое регистра адреса заменяется на 11 младших разрядов регистра данных (рис. 2.9,*а*), т. е. на адрес адреса операнда, а затем в регистр данных считывается сам адрес операнда (рис. 2.9,*б*).

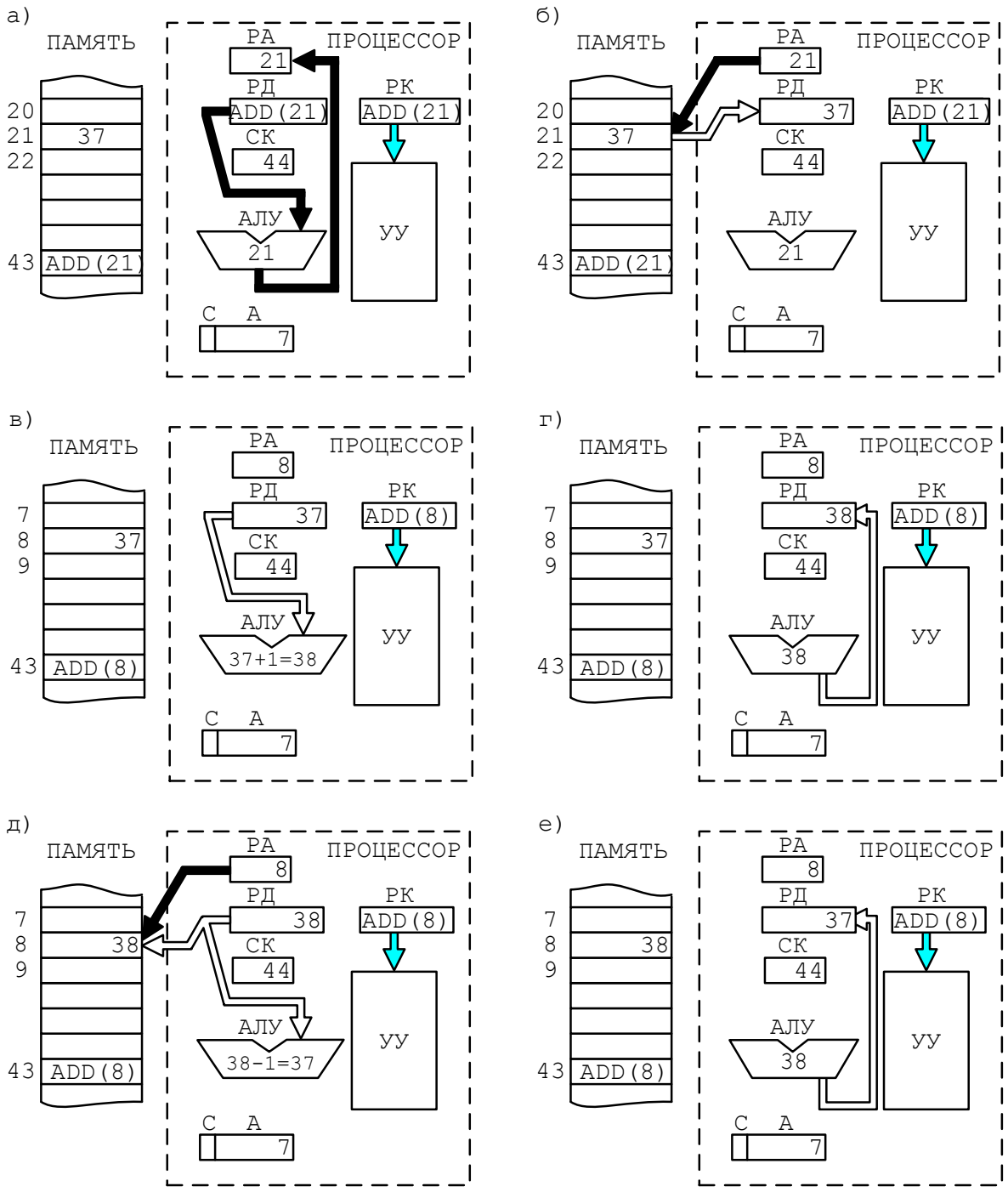


Рис. 2.9. Цикл выборки адреса

Если косвенно адресуется одна из индексных ячеек (ячеек с номерами от 008 до 00F), то цикл выборки адреса операнда (результата) продолжается:

3) содержимое регистра данных увеличивается на единицу (рис. 2.9,*в,г*);

4) измененное содержимое регистра данных, пересылается в ячейку памяти по адресу, указываемому регистром адреса (рис. 2.9,*д*);

5) содержимое регистра данных уменьшается на 1 (рис. 2.9,*д,е*).

После этой операции в регистре данных восстанавливается значение адреса, находившегося в индексной ячейке до выполнения шага 3. Содержимое же индексной ячейки увеличилось на единицу, и при следующем обращении к ней будет выбран новый адрес операнда (результата).

Для иллюстрации шагов 3, 4 и 5 выбрана команда ADD (8), также расположенная в ячейке 43. Поэтому два первых шага по выборке адреса операнда (результата) будут очень похожи на аналогичные действия при выполнении команды ADD (21), которые показаны на рис. 2.9,*а,б*. Различие лишь в том, что в регистрах данных и команд первоначально сохранялась команда ADD (8) вместо ADD (21) и, следовательно, регистр адреса указывает на ячейку 8 (а не на ячейку 21).

Исполнение. Последовательность действий, выполняемых в этом цикле, определяется типом выполняемой адресной команды.

А. Для команд, при выполнении которых требуется выборка операнда из памяти ЭВМ (AND, ADD, ADC, SUB, ISZ), состояние исполнения используется для чтения операнда в регистр данных и выполнения операции, указываемой кодом операции команды.

Пример цикла исполнения для команды ADD 21 подробно рассмотрен на рис. 2.2,*д* - 2.2,*з*. И хотя на рис. 2.2 иллюстрировалось выполнение команды сложения с прямой адресацией, оба цикла (выборки и исполнения) будут совершенно такими же и при сложении с косвенной адресацией. В последнем случае между этими циклами будет выполнен цикл выборки адреса операнда, и этот адрес, так же, как и при прямой адресации, будет помещен в регистр данных.

Б. По команде пересылки (MOV) в этом машинном цикле производится запись содержимого аккумулятора в ячейку памяти с адресом, расположенным в регистре данных, для чего содержимое регистра данных пересылается в регистр адреса, а содержимое аккумулятора - в регистр данных и далее в ячейку памяти, указываемую регистром адреса.

В. При исполнении команд переходов (BCS, BPL, BMI, BEQ) производятся проверка соответствующего условия (1 в регистре переноса, 0 в знаковом разряде аккумулятора и т. п.) и пересылка адреса из регистра данных в счетчик команд при выполнении этого условия. Если исполняется команда безусловного перехода (BR), то пересылка адреса перехода в счетчик команд осуществляется без какой-либо проверки.

Следовательно, при выполнении условия, определяемого кодом операции команды, или при выполнении команды BR следующей будет выбираться команда из ячейки памяти с адресом, расположенным в регистре данных (адресом, расположенным в исполняемой команде или выбранным из ячейки, на которую указывает этот адрес). В противном случае будет выбрана команда, расположенная вслед за исполняемой.

Г. Для команды обращения к подпрограмме (JSR) во время этого машинного цикла осуществляются пересылка содержимого счетчика команд в ячейку памяти, адрес которой содержится в регистре данных, и занесения в счетчик команд увеличенного на единицу содержимого регистра данных.

Эти действия иллюстрируются на примере выполнения команды JSR 30 (рис. 2.10). Здесь в качестве регистра для временного хранения адреса первой команды подпрограммы служит регистр команд, содержимое которого после декодирования команды уже не используется устройством управления.

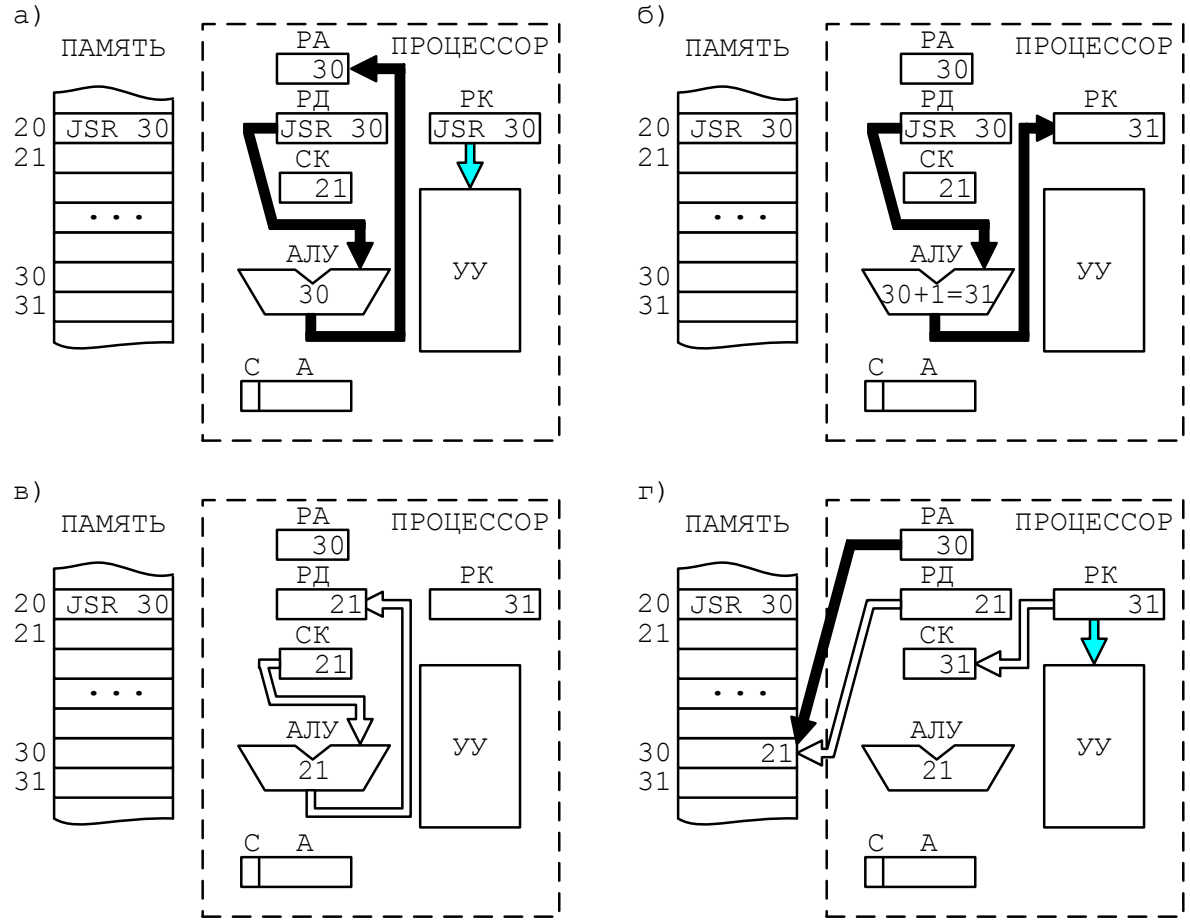


Рис. 2.10. Цикл "Исполнение" команды JSR 30

Сначала адресная часть команды пересылается из регистра данных в регистр адреса (рис. 2.10,*а*). Затем этот же адрес наращивается на единицу (формируется адрес первой команды подпрограммы) и пересылается на временное хранение в регистр команд (рис. 2.10,*б*). Потом содержимое счетчика команд (адрес команды, которая должна выполняться после возврата из подпрограммы) пересылается в регистр данных (рис. 2.10,*в*) и далее в ячейку памяти, указываемую регистром адреса (рис. 2.10,*г*). Наконец, в счетчик команд переписывается из регистра команд адрес первой команды подпрограммы (рис. 2.10,*г*).

Таким образом организуются переход к выполнению команд подпрограммы (начиная с команды в ячейке 31) и запоминание (перед первой ее командой) адреса возврата из подпрограммы. Когда в конце подпрограммы будет выполнена команда BR (30), то в счетчик команд попадет адрес 21, сохраняемый в ячейке 30, и машина продолжит выполнение команд, расположенных за командой JSR 30.

Как было указано в начале параграфа, время отладки машинных программ можно сократить, если знать, как реализуются команды ЭВМ, и иметь возможность выполнять отлаживаемую программу по отдельным командам, а иногда и по машинным циклам. Такая возможность предоставлена пользователю базовой ЭВМ. Он может выполнять программу не только по командам и машинным циклам, но и по отдельным тактам, т. е. проследить процесс выполнения каждой из команд, узнать, какие регистры использует процессор при их выполнении, какая информация размещается в этих регистрах и как реагирует микропрограммное устройство управления на изменение содержимого тех или иных регистров.

С процессом отладки можно ознакомиться в параграфе 3.1, где дано описание отладочного пульта базовой ЭВМ.

3. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА ИНФОРМАЦИИ В БАЗОВОЙ ЭВМ

3.1. Отладочный пульт базовой ЭВМ

К ЭВМ можно подключать большое число разнообразных устройств ввода-вывода или, как часто говорят, внешних устройств (ВУ). В гл. 8 и последующих параграфах данной главы рассмотрена организация связей между процессором и ВУ. Здесь же речь пойдет о простейшем из таких устройств - отладочном пульте. На рис. 3.1 и 4.7 изображены реально существовавшие отладочные пульты, которые некоторое время назад стояли в лаборатории и подключались к моделям базовой ЭВМ. В настоящее время изучение работы базовой ЭВМ производится с помощью эмулятора. Версии эмуляторов периодически меняются, и каждая из них по-своему отображает на экране отладочный пульт. Поэтому приведенное ниже описание пульта отличается от того, что отображается на экране при работе эмулятора. Вместо индикаторов с линейкой загорающихся лампочек на экран выводятся единицы и нули, а для ввода данных вместо панели тумблеров и кнопок используется клавиатура. Однако, описание пульта оставлено без изменений, чтобы каждый раз при появлении новой версии эмулятора не приходилось его переписывать. А с помощью подсказки на экране эмулятора можно узнать, как та или иная операция пульта реализована в конкретной версии.

Отладочный пульт базовой ЭВМ (рис. 3.1) позволяет произвести ввод программы и данных, подробно ознакомиться с процессом выполнения этой программы, рассмотреть взаимодействие процессора с несколькими внешними устройствами и т. п.

На пульт выведены индикаторы состояния основных регистров машины, 16 ячеек памяти (среди которых находится адресуемая в текущий момент ячейка), двух устройств управления (контроллеров) устройств ввода информации и одного контроллера устройства вывода информации, а также индикаторы состояний устройства управления базовой ЭВМ.

Индикатор каждого регистра или ячейки памяти представляет собой горизонтально расположенную линейку лампочек, число которых соответствует числу разрядов в регистре (ячейке памяти). Единичное состояние разряда отображается горящей лампочкой: если содержимое разряда равно нулю, соответствующая лампочка не горит. Для удобства считывания содержимого регистра (ячейки памяти) их разряды объединяются в группы по тетрадам (эквивалентам шестнадцатеричных цифр).

Под индикаторами расположена панель с тумблерами и кнопками, ис-пользуемыми для ввода команд и данных, а также управления работой машины. Рассмотрим, как с их помощью можно производить ввод и

проверку программы, обеспечить ее выполнение по машинным циклам, командам или в автоматическом режиме и т. п.

Ручные операции. Все ручные операции выполняются только тогда, когда тумблер РАБОТА/ОСТАНОВ находится в положении ОСТАНОВ.

Ввод программы и (или) данных. При этом осуществляются следующие шаги:

1) набрать на клавишном регистре (АДРЕС/ДАННЫЕ) адрес ячейки, в которую надо занести команду или число (для набора единицы тумблер должен быть установлен в положение от себя, а для набора нуля - в противоположное положение);

2) нажать кнопку ВВОД АДРЕСА (это приведет к переписи содержимого клавишного регистра в счетчик команд);

3) набрать на клавишном регистре код команды или числа;

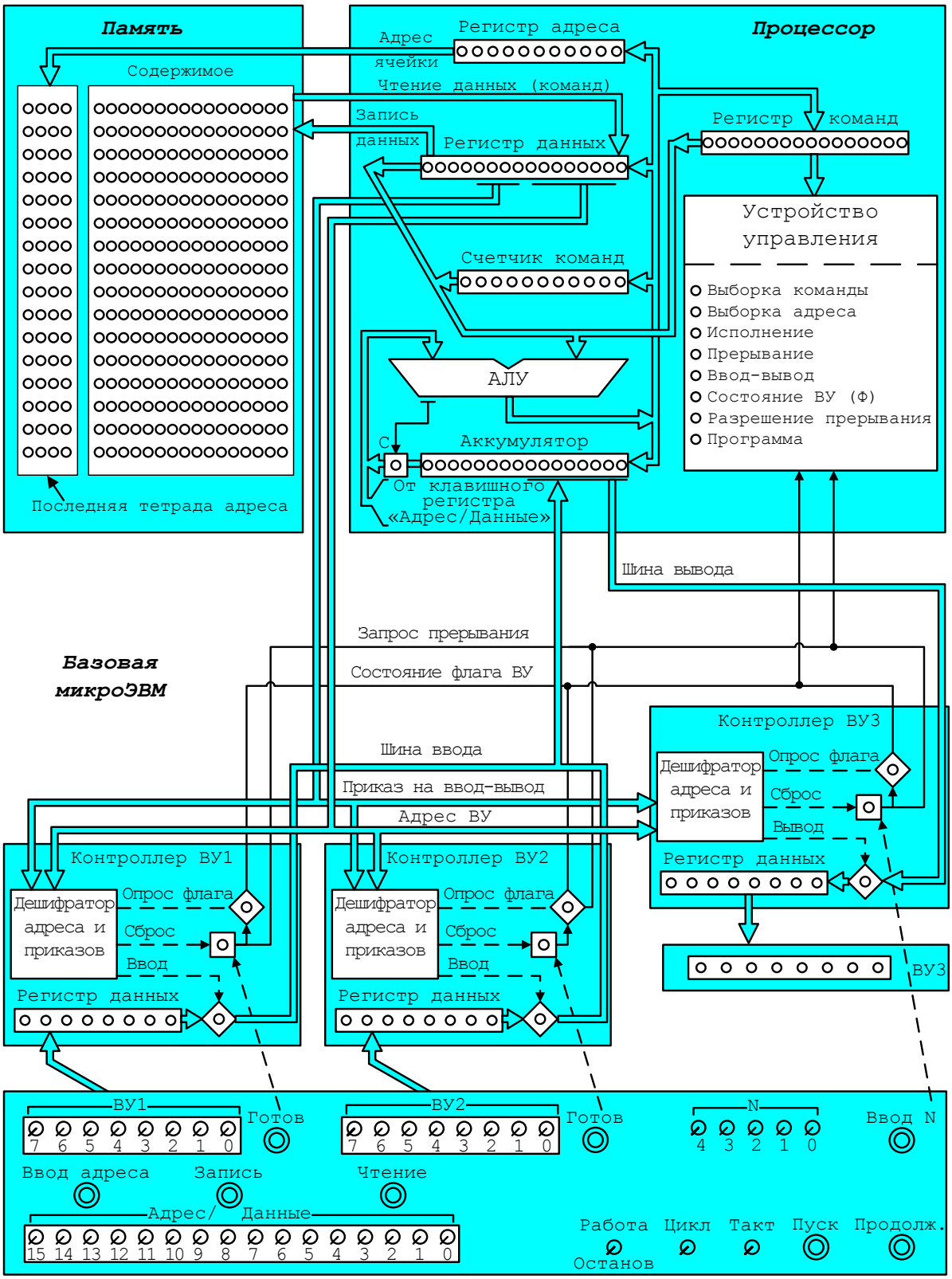


Рис. 3.1. Отладочный пульт базовой ЭВМ

4) нажать кнопку ЗАПИСЬ (это приведет к переписи содержимого счетчика команд в регистр адреса, а содержимого клавишного регистра - в

регистр данных и далее в ячейку памяти, на которую указывает регистр адреса).

Так как в момент записи содержимое счетчика команд автоматически увеличивается на единицу, то для записи команд или данных в последовательные ячейки памяти требуется лишь выполнение пп. 3 и 4. Для записи же в разрозненные ячейки памяти каждый раз надо выполнять все четыре шага.

Чтение программы и (или) данных. Два первых шага такие же, как при вводе команд (данных). Затем нажимается кнопка ЧТЕНИЕ, что приводит к переписи в регистр данных содержимого ячейки, на которую указывает регистр адреса, и к наращиванию на единицу содержимого счетчика команд. Таким образом, для просмотра содержимого следующей по порядку ячейки достаточно лишь снова нажать кнопку ЧТЕНИЕ.

Подготовка к исполнению программы. Перед подачей приказа на выполнение программы, введенной в память ЭВМ, в счетчик команд должен быть занесен адрес первой исполняемой команды этой программы (пусковой адрес). Для этого тумблер РАБОТА/ОСТАНОВ переводится в положение ОСТАНОВ, на клавишном регистре набирается пусковой адрес программы и нажимается кнопка ВВОД АДРЕСА.

Затем производится операция по установке требуемого режима выполнения программы: автоматического (команда за командой с остановом после выполнения команды HLT или по приказу с отладочного пульта) или пошагового (по отдельным командам, машинным циклам или тактам). Автоматический режим подготавливается переводом тумблера РАБОТА/ ОСТАНОВ в положение РАБОТА. Для подготовки пошагового режима надо перевести этот тумблер в положение ОСТАНОВ и установить должным образом тумблеры ЦИКЛ и (или) ТАКТ. Выключение тумблеров (на себя) подготавливает режим работы по командам, включение тумблера ЦИКЛ - по машинным циклам, включение тумблера ТАКТ - по тактам, одновременное включение тумблеров ЦИКЛ и ТАКТ - по тактам.

Исполнение программы. Исполнение всей программы (подготовлен автоматический режим) можно инициировать двумя способами: нажатием кнопки ПУСК или кнопки ПРОДОЛЖЕНИЕ.

При нажатии кнопки ПУСК сначала очищаются (устанавливаются в нуль) аккумулятор, регистры адреса, данных, команд, переноса и флаги ВУ, устанавливаются состояние запрещения прерывания, режим ПРОГРАММА, начало цикла "Выборка команды" и начинается последовательное выполнение команд программы.

При нажатии кнопки ПРОДОЛЖЕНИЕ устанавливается режим ПРОГРАММА и начинается выполнение программы с той команды (того машинного цикла и его такта), на которой было прервано ее выполнение. Если до этого осуществлялись операции по занесению пускового адреса

программы, то выполнение начнется с выборки команды, расположенной по этому адресу, при использовании "старых" (не равных нулю) содержимого основных регистров ЭВМ.

Так как во всех программах, рассмотренных в табл. 2.1, 2.7, 2.8 и примерах 2.4-2.6, пусковая команда (CLA) обнуляет аккумулятор, то выполнение этих программ можно с равным успехом инициировать как кнопкой ПУСК, так и кнопкой ПРОДОЛЖЕНИЕ.

Перед исполнением программы по командам (машинным циклам и/или тактам) может быть нажата кнопка ПУСК. При этом производится обнуление аккумулятора, регистров адреса, данных, команд, переноса и флагов ВУ, устанавливаются состояние запрещения прерывания и начало цикла "Выборка команды".

Последовательное выполнение отдельных команд или машинных циклов (в зависимости от подготовленного ранее режима) инициируется нажатием кнопки ПРОДОЛЖЕНИЕ.

Если установлен режим ТАКТ, то нажатие кнопки ПРОДОЛЖЕНИЕ приводит к выполнению лишь первого такта по переходу к выборке команды, после чего ЭВМ не реагирует на нажатие этой кнопки, но зато реагирует на нажатие кнопки ВВОД N. С помощью последней можно исполнить все микрокоманды реализуемой команды, и когда будет исполнена последняя, ЭВМ перестанет реагировать на нажатие кнопки ВВОД N. Для выполнения по тактам следующей команды программы надо опять нажать один раз кнопку ПРОДОЛЖЕНИЕ, а затем нажимать кнопку ВВОД N до тех пор, пока ЭВМ не перестанет реагировать на эти приказы.

Из пошагового режима выполнения программы в любой момент можно перейти к ее автоматическому выполнению. Для этого тумблер РАБОТА/ ОСТАНОВ переводится в положение РАБОТА и нажимается кнопка ПРОДОЛЖЕНИЕ.

Переход из автоматического режима в пошаговый осуществляется с помощью переключения тумблера РАБОТА/ОСТАНОВ в положение ОСТАНОВ. При этом ЭВМ заканчивает выполнение всех циклов текущей команды и останавливается. Теперь после просмотра на индикаторах пульта состояния основных устройств ЭВМ можно продолжить выполнение программы либо в автоматическом режиме (вернуть тумблер РАБОТА/ОСТАНОВ в положение РАБОТА и нажать кнопку ПРОДОЛЖЕНИЕ), либо по командам, машинным циклам или тактам.

Отладка программ. Выше (см. параграф 2.5) говорилось, что возможность пошагового выполнения программы позволяет ускорить процесс ее отладки. Для иллюстрации этого утверждения рассмотрим отладку программы в примере 2.5. Эта программа предназначена для умножения двух чисел (в примере сомножители равны 5 и $120=(78)_{16}$). В табл. 3.1 программа представлена в символической (как в примере 2.5), шестнадцатеричной и двоичной формах, причем последние две –

правильная и ошибочная (вместо команды BR 1 записана команда BR 0 и вместо числа -5 – число -3).

Таблица 3.1

Программа умножения двух чисел

Адрес	Содержимое			
	Мнемоническая форма	Шестнадцатеричная форма	Двоичная форма	
			Правильно	С ошибкой
0	CLA	F200	1111 0010 0000 0000	1111 0010 0000 0000
1	ADD 10	4010	0100 0000 0001 0000	0100 0000 0001 0000
2	ISZ 11	0011	0000 0000 0001 0001	0000 0000 0001 0001
3	BR 1	C001	1100 0000 0000 0001	1100 0000 0000 0000
4	HLT	F000	1111 0000 0000 0000	1111 0000 0000 0000
…	…	…	…	…
10		0078	0000 0000 0111 1000	0000 0000 0111 1000
11		FFFB	1111 1111 1111 1011	1111 1111 1111 1101

После выполнения программы в автоматическом режиме аккумулятор (в котором должно располагаться произведение 120x5=600=$(258)_{16}$) содержал число (0000 0000 0111 1000) 2 = $(78)_{16}$= $(120)_{10}$. Создалось впечатление, что в ячейке 11 вместо числа -5 записано число - 1. Однако в процессе выполнения программы к содержимому этой ячейки последовательно добавлялись единицы, и в конце ее содержимое стало равно нулю. Следовательно, теперь уже не установить первоначального содержимого ячейки 11.

Повторное выполнение программы (после восстановления содержимого ячейки 11) дало тот же результат, и было принято решение выполнять программу по командам, занося результаты ее выполнения в табл. 3.2.

В процессе выполнения команд выясняется, что в ячейку 11 было занесено число -3 вместо числа -5. Однако это не могло дать результата, равного (78) 16, так как (78) 16 X 3 # (78) ie. Поэтому продолжаем поиск ошибки.

Таблица 3.2

Выполнение команд программы умножения двух чисел

Выполняемая команда		Содержимое регистров после выполнения команды						Комментарии
Адрес	Код	СК	РА	РК	РД	С	А	
000	F200	0001	0000	F200	F200	0	0000	0=>A
001	4010	0002	0010	4010	0078	0	0078	$(78)_{16}$=>A
002	0011	0003	0011	0011	FFFE	0	0078	В РД должно быть -4, а не -2, появившееся из-за того, что в ячейку 11
003	C000	0000	0003	C000	C000	0	0078	было занесено -3
000	F200	0001	0000	F200	F200	0	0000	Переход по адресу 0000 Аккумулятор очищен из-за ошибочного перехода на команду CLA

После окончания первого цикла выясняется, что вместо перехода на суммирование содержимого аккумулятора и ячейки 11 произведено обнуление аккумулятора. Теперь становится ясным, почему результат был равен $(78)_{16}$. Сколько бы циклов ни содержала программа, сумма, полученная в каждом из них (кроме последнего), будет затираться.

Могут возникнуть вопросы: "Надо ли было проверять программу путем последовательного выполнения команд? Не лучше ли просто внимательно проверить коды этих команд?" Если пользователь имеет правильный вариант кодировки, то тщательная проверка содержимого памяти помогает нахождению ошибок. Однако при большом объеме программы, когда приходится проверять сотни и тысячи 0 и 1, довольно быстро притупляется внимание и ошибки пропускаются. Кроме того, есть еще ошибки, которые возникли на этапе программирования: коды команд в памяти ЭВМ и на бумаге совпадают, но не соответствуют заданному алгоритму. Эта ситуация обнаруживается лишь при выполнении программы на ЭВМ.

3.2. Устройства ввода-вывода базовой ЭВМ

При разработке внешних устройств редко ориентируются на их подключение к определенному типу ЭВМ. Очень быстрое развитие вычислительной техники и, следовательно, частое обновление парка ЭВМ привели бы в этом случае к необходимости изменения тех схем ВУ, которые связывают его с конкретной ЭВМ, т. е. к изменению технологического процесса изготовления ВУ. Затраты на изготовление ВУ, пригодных для использования с разными типами ЭВМ (даже с теми, которые еще не разрабатывались), существенно уменьшаются, если при разработке ВУ ориентироваться только на лучшее выполнение ими заданных функций (печати, воспроизведения звука и т.п.). При этом для связи ВУ с ЭВМ можно использовать специальные управляющие устройства - контроллеры (англ. controller - управляющий, ревизор). Тогда для подключения ВУ к новому типу ЭВМ потребуется лишь разработать и изготовить новый контроллер.

В базовой ЭВМ, которая создавалась для обучения принципам функцио-нирования ЭВМ, используются простейшие ВУ: два устройства ввода (ВУ-1 и ВУ-2) и одно устройство вывода (ВУ-3), размещенные на отладочном пульте (рис. 3.1). Устройство ввода представляет собой 8-разрядный тумблерный регистр и кнопку (ГОТОВ) для выдачи сигнала о том, что информация, набранная с помощью тумблеров, может быть считана в ЭВМ. Устройство вывода представлено 8-разрядным регистром, состояние которого высвечивается на отладочном пульте, и кнопкой (ВВОД N) для выдачи сигнала о том, что информация может быть принята в этот регистр.

Между ВУ и процессором установлены простейшие контроллеры, каждый из которых содержит:

- регистр данных (для обмена данными между ВУ и процессором);
- дешифратор адреса (позволяющий выделить обращение к данному ВУ среди всех обращений к устройствам ввода-вывода, подключенным к процессору);
- дешифратор приказов (декодирующий приказ от процессора на выполнение тех или иных операций);
- регистр состояния (в котором хранится информация о готовности ВУ к обмену данными с процессором).

В контроллерах простейших ВУ обычно используются однобитовые регистры состояния, которые часто называют флагом или флажком. Это название распространяется и на контроллеры базовой ЭВМ.

Контроллеры связаны с процессором шинами ввода и вывода информации, шиной адреса и шиной управления, по которой передаются приказы от процессора в контроллеры ВУ и сведения о состоянии ВУ в процессор.

3.3. Программно-управляемая передача информации

Устройства ввода-вывода передают в ЭВМ и получают из нее, как правило, большой объем информации, который не может быть размещен только в регистрах процессора. Поэтому информация передается из ВУ в память ЭВМ и поступает на ВУ из ее памяти. При этом обмен может идти под управлением программы ЭВМ через регистры процессора (программно-управляемая передача данных) или под управлением специального внешнего устройства (контроллера прямого доступа в память), минуя процессор (передача данных при прямом доступе к памяти). Программно-управляемый обмен осуществляется малыми порциями, при прямом доступе к памяти информация передается большими блоками.

В базовой ЭВМ реализована только программно-управляемая передача данных.

1. Операция вывода предусматривает пересылку информации из аккумулятора процессора в регистр данных контроллера ВУ. После того как данные приняты этим регистром, начинают выполняться операции по их преобразованию в связанном с контроллером ВУ (например, печать соответствующего символа или изменение положения исполнительного органа). В базовой ЭВМ просто изменяется состояние индикатора регистра вывода ВУ-3.

2. Операция ввода предусматривает передачу данных из регистра данных контроллера ВУ в аккумулятор ЭВМ. В базовой ЭВМ устройство ввода имитируется с помощью набора тумблеров, состояние которых считывается в регистр данных при исполнении команды ввода.

При использовании программно-управляемого обмена должна быть составлена программа для пересылки данных из памяти ЭВМ в аккумулятор и далее в регистр данных контроллера ВУ (или из регистра данных контроллера ВУ в аккумулятор и затем в память ЭВМ).

В этой программе можно реализовать один из трех видов подобного обмена: синхронный, асинхронный и по прерыванию.

Синхронный обмен можно использовать лишь для связи с такими ВУ, для которых точно известно время выполнения операции (например, максимально возможное время, затрачиваемое на печать одного символа и т. п.). При этом программу обмена необходимо составлять так, чтобы команды на обмен шли с интервалами не меньшими, чем время выполнения одной операции обмена.

Это наиболее простой вид обмена, требующий минимальных затрат технических и программных средств. Однако при его использовании для передачи данных между ЭВМ и сравнительно медленными ВУ, как правило, не удается полностью загрузить ЭВМ на время между двумя пересылками данных. Так, например, электрическая пишущая машинка позволяет организовать вывод данных со скоростью не выше 7 символов в секунду, а за 1/7 секунды микроЭВМ может выполнить от 10 000 до 1 000 000 операций.

Асинхронный обмен широко используется в микроЭВМ. При его реализации интервал между командами передачи данных задается самим внешним устройством. Контроллеры этих устройств снабжаются регистром состояния, который информирует ЭВМ о готовности устройства к обмену информацией (например, электрическая пишущая машинка устанавливает сигнал готовности лишь тогда, когда она закончила печать предыдущего символа, в нее заправлены бумага и красящая лента). Программа обмена строится так.

1. Проверяется состояние устройства ввода-вывода.
2. Если устройство готово к обмену, то производится переход к п. 3. В противном случае вновь выполняется п. 1.
3. Производится передача очередной порции данных, выполняются действия по их обработке и (или) другие операции, и если обмен данными еще не закончен, осуществляется переход к п. 1.

Таким образом, при асинхронном обмене программист не должен знать о времени выполнения операции на ВУ. Последнее само информирует ЭВМ о завершении предыдущей операции и готовности к выполнению новой операции. Однако ЭВМ должна тратить время на ожидание момента готовности, а так как готовность проверяется программным путем (команда TSF B), то в это время ЭВМ не может выполнять никакой другой работы по преобразованию данных.

Обмен по прерыванию программы отличается от асинхронного обмена тем, что готовность ВУ к обмену проверяется при помощи

аппаратных средств ЭВМ, а не программным путем. Это позволяет организовать работу ЭВМ следующим образом.

Машина исполняет какую-либо программу (назовем ее основной или фоновой), не связанную с обменом. Когда ВУ готово к приему или выдаче информации, оно посылает в ЭВМ сигнал готовности, воспринимаемый специальным блоком ЭВМ (например, контроллером прерываний). Этот блок приостанавливает (прерывает) исполнение основной программы и передает управление подпрограмме, организующей нужный вид обмена данными. Когда выполнение подпрограммы завершается, возобновляется работа ЭВМ по основной (временно прерванной) программе.

Различие между рассмотренными видами программно-управляемого обмена можно проиллюстрировать на следующем примере.

Пример 3.1. Вам поручено вскипятить воду в чайнике.

1. Безусловная операция (синхронный обмен). В 8 ч Вы поставили на плиту чайник с водой и, зная, что обычно он закипает через 10 мин, ровно в 8 ч 10 мин сняли его с плиты не заботясь о том, закипела вода или нет.

2. Условная операция (асинхронный обмен). Вы поставили на плиту чайник с водой и примерно раз в минуту посматриваете за ним. Когда чайник закипит, Вы снимете его с плиты.

3. Операция по прерыванию. Вы поставили на плиту чайник с водой, надели на его носик свисток и стали читать книгу (фоновая работа). Когда засвистит свисток, вы снимете чайник с плиты и продолжите чтение книги.

В параграфах 3.4 и 3.5 рассмотрены наиболее употребительные виды обмена: асинхронный и по прерыванию программы.

Команды ввода-вывода базовой ЭВМ имеют одинаковый формат (см. рис. 2.2,в): 4-разрядное поле кода операции, 4-разрядное поле кода приказа и 8-разрядное поле кода выборки устройства ввода-вывода.

Код операции $(1110)_2$ обрабатывается в процессоре ЭВМ и служит для отличия команд ввода-вывода от других команд машины.

Код приказа используется для передачи контроллеру ВУ приказа на выполнение того или иного действия. Так как для этого кода отведены четыре разряда, то ЭВМ может переслать 16 различных команд внешнему устройству. Декодирование приказа осуществляется контроллером ВУ.

В базовой ЭВМ задаются три основных приказа: пересылка данных (IN B – ввод и OUT B – вывод), проверка готовности ВУ (TSF B) и сброс состояния готовности (CLF B), где B – адрес ВУ.

Код выборки устройства (адрес ВУ) используется для организации связи ЭВМ с одним из подключенных к ней внешних устройств. Этот код одновременно передается на все ВУ (их может быть 28 = 256) и анализируется в их контроллерах. Когда последний распознает код, принадлежащий данному ВУ, он связывает ВУ с ЭВМ. Все другие устройства не будут связаны с ЭВМ и не реагируют на ее приказы.

В командах ввода-вывода (см. табл. 2.4) используется слово флаг. Так часто называют однобитовый регистр состояний ВУ, который

устанавливается в единичное состояние, когда устройство готово к обмену информацией. Когда флажок ВУ сброшен (установка в нуль), ВУ занято: устройство вывода еще обрабатывает предыдущую команду, а устройство ввода не закончило подготовку данных.

Команда CLF B (E0xx, где xx - две шестнадцатеричные цифры кода выборки ВУ) служит для установки в исходное состояние ВУ с кодом выборки B. Команда CLF 02 сбрасывает (устанавливает в нуль) флажок в контроллере ВУ с кодом выборки 02.

По команде TSF B (Elxx) в РЕГИСТР СОСТОЯНИЙ ВУ (Ф в устройстве управления на рис. 3.1) помещается содержимое флажка ВУ с кодом выборки B. Затем процессор суммирует содержимое регистра Ф с содержимым счетчика команд. Следовательно:

при Ф = 0 (устройство B не готово к обмену) выполняется следующая за TSF B команда программы;

при Ф = 1 (устройство B готово к обмену) следующая за TSF B команда пропускается и выполняется команда, расположенная через одну за TSF B.

Команда IN B (E2xx) служит для пересылки в восемь младших разрядов аккумулятора содержимого регистра данных контроллера ВУ с кодом выборки B.

Команда OUT B (E3xx) служит для пересылки содержимого восьми младших разрядов аккумулятора в регистр данных устройства с кодом выборки B.

Когда в процессе работы ЭВМ устройство управления обнаруживает, что в регистре команд и одновременно в регистре данных находится команда с кодом операции $(1110)_2$, производится переход в режим ВВОД-ВЫВОД и на все контроллеры ВУ передается содержимое 12 младших разрядов регистра данных (см. рис. 3.1). Дешифраторы адреса всех контроллеров ВУ декодируют это содержимое, но срабатывает тот из них, код (адрес) которого совпадает с адресом, установленным в команде.

Этот дешифратор открывает вентили (см. параграф 1.2) для передачи информации на дешифратор приказов, который декодирует содержимое 8-11-го разрядов регистра данных и выдает приказ на выполнение одной из перечисленных выше команд: открывает вентиль для передачи в ЭВМ состояния флага (TSF), обнуляет флаг (CLF) или открывает вентили, связывающие аккумулятор с регистром данных контроллера (IN или OUT).

Для передачи в КОНТРОЛЛЕР ПРЕРЫВАНИЙ состояния флажков всех ВУ используется линия "Запрос прерывания", а для передачи в РЕГИСТР СОСТОЯНИЙ ВУ (Ф) состояния флажка опрашиваемого ВУ – линия "Состояние флага". Данные обычно передаются по одной шине данных, которая в базовой ЭВМ (для упрощения описания) представлена двумя шинами: шиной ввода и шиной вывода.

Программно-управляемый обмен осуществляется малыми порциями – обычно байтами. Такая 8-разрядная единица информации позволяет закодировать до 256 (2^8) различных символов, передаваемых между ЭВМ и ВУ: цифры, буквы латинского и русского алфавитов, знаки математического действия и т. п. (см. Приложение Б). Поэтому шины ввода и вывода базовой ЭВМ связывают 8-разрядные регистры данных контроллеров ВУ с восьмью младшими разрядами аккумулятора.

Так как слово базовой ЭВМ имеет 16 разрядов, то в одном таком слове можно разместить для хранения коды двух символов. В связи с этим программы организации обмена между памятью ЭВМ и различными ВУ целесообразно составлять следующим образом.

1. При вводе первый символ, который попал в младшие разряды аккумулятора, сдвигается в его старшие разряды. Затем в младшие разряды аккумулятора принимается следующий символ, и только тогда производится пересылка содержимого аккумулятора (кодов двух символов) в память.
2. При выводе в аккумулятор пересылается из памяти слово, содержащее два выводимых символа. Затем на ВУ выводится символ, код которого расположен в младших разрядах аккумулятора, и после сдвига – второй символ.

3.4. Асинхронный обмен

При асинхронной передаче данных необходимо сначала убедиться, что ВУ готово к обмену (включено и установлено в исходное состояние или уже закончило выполнение предыдущей операции), и лишь тогда давать команду на обмен (ввод или вывод).

О своей готовности к обмену ВУ сообщает с помощью установки в единичное состояние флага в контроллере ВУ. Для выяснения состояния этого флага следует выполнить команду "Опрос флага" (TSF), и тогда по линии "Состояние флага" в регистр состояний ВУ будет передано содержимое флага того устройства, адрес которого указан в команде опроса (см. рис. 3.1). В зависимости от этого содержимого (0 или 1) будет соответственно выполняться либо следующая команда программы, либо команда, расположенная за следующей (следующая команда будет пропущена).

Пример 3.2. С помощью ВУ-1 записать в ячейку 006 коды символов слова "Да".

Программа для выполнения этого задания приведена в табл. 3.3. При работе по такой программе ЭВМ будет ожидать нажатия кнопки ГОТОВ (имитирующей появления готовности ВУ к обмену информацией). Поэтому до первого нажатия этой кнопки необходимо набрать на тумблерах ВУ-1 код символа Д, затем нажать кнопку и приступить к набору кода символа А. В процессе этого набора ЭВМ занята сбросом флага готовности ВУ, пересылкой кода символа Д в младшие разряды аккумулятора и сдвигом этого кода в старшие разряды, чтобы подготовиться к приему кода следующего символа.

Так как быстродействие ЭВМ во много раз превышает быстродействие человека, то к моменту окончания набора кода символа А машина не только закончит действия по обработке символа Д, но и тысячи раз проверит содержимое флага ВУ-1. В связи с этим после набора кода можно сразу же нажимать кнопку ГОТОВ, т. е. давать команду на пересылку этого кода в аккумулятор ЭВМ. Теперь в аккумуляторе находится код слова "Да" и его содержимое можно пересылать в заданную ячейку.

Здесь следует напомнить, что состояние флагов подключенных к ЭВМ контроллеров ВУ пересылается в процессор по двум линиям: "Запрос прерывания" и "Состояние флага" (см. рис. 3.1). Линия "Запрос прерывания" непосредственно связана со всеми флагами контроллеров ВУ, а линия "Состояние флага" - через вентили, которые открываются лишь по командам "Опрос флага" (TSF). При асинхронном обмене используется линия "Состояние флага". Поэтому нажатие любой другой кнопки ГОТОВ, кроме той, которая относится к устройству с адресом, указанным в команде TSF, не приведет к выполнению действий по вводу (или выводу) информации, хотя на линии "Запрос прерывания" и появится сигнал готовности.

Таблица 3.3

Программа ввода кода двух символов

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
5	FFF8		Константа -8, используется для сдвига
6	0000		Ячейка для записи «Да»
	…		
20	E101	TSF 1	Опрос флага контроллера ВУ-1 и повторение этой
21	C020	BR 20	операции, если ВУ не готово к обмену (флаг=0)
22	E201	IN 1	Это действие осуществляется лишь после нажатия кнопки ГОТОВ у ВУ-1, т.е. когда при выполнении TSF 1 выясняется, что флаг=1 и пропускается BR 20. По коман-де IN 1 содержимое регистра данных контроллера ВУ-1 пересылается в восемь младших
23	E001	CLF 1	разрядов аккумулятора
			Сброс готовности ВУ-1 для предотвращения считывания кода с ВУ-1 до тех пор, пока на нем не будет установлен код символа А и об этом не
24	F600	ROL	будет сообщено нажатием кнопки ГОТОВ
25	0005		Код первого введенного символа (Д) сдвигается на
26	C024		восемь разрядов влево, освобождая место для
27	E101		ввода следующего символа
28	C027		Опрос флага контроллера ВУ-1 … (см.
29	E201		комментарий к командам 20 и 21
			Ввод кода символа, установленного на тумблерах
2A	E001		ВУ-1 (если подан сигнал готовности ВУ-1)
2B	3006		Сброс готовности ВУ-1
2C	F000	HLT	Пересылка «ДА» в ячейку 006
			Останов ЭВМ

3.5. Обмен по прерыванию программы

Передача данных с прерыванием программы особенно удобна при работе с низкоскоростными ВУ (пишущими машинками, перфораторами, телетайпами и т. п.), а также в ситуациях, когда момент передачи данных в ЭВМ непредсказуем, например при работе с каналами связи. Основной характерной чертой рассматриваемой передачи является то, что обмен данными между ЭВМ и ВУ инициируется сигналом с ВУ. Для реализации данного типа обмена необходимо заменить программный цикл ожидания готовности ВУ при асинхронной передаче (TSF, BR) аппаратной проверкой наличия внешнего прерывания, т. е. сигнала готовности по линии "Запрос прерывания" (см. рис. 3.1).

В этом случае ЭВМ может выполнять любую программу (будем называть ее основной), а когда с ВУ поступит запрос прерывания (запрос на передачу или прием данных), приостановить (прервать) выполнение этой программы на время осуществления обмена данными. Все действия

по приостановке программы и переходу к подпрограмме обмена реализуются в базовой ЭВМ с помощью контроллера прерываний, входящего в состав устройства управления (см. рис. 3.1).

По командам "Разрешение прерывания" (E1) или "Запрещение прерывания" (D1) контроллер прерываний переходит в одно из двух состояний, в которых он соответственно реагирует или не реагирует на сигналы готовности ВУ, передаваемые по линии "Запрос прерывания". Если контроллер установлен в состояние разрешения прерывания, то последовательность шагов по осуществлению рассматриваемой передачи данных обычно имеет такой вид.

Шаг 1. По завершении выполнения текущей команды основной программы управление передается контроллеру прерывания. Если в этот момент на линии "Запрос прерывания" нет сигнала о единичном состоянии флага какого-либо из ВУ, то начинаются выборка и исполнение следующей команды основной программы и данный шаг повторяется. При наличии запроса прерывания от ВУ выполняется второй шаг.

Шаг 2. Контроллер прерываний переходит в состояние запрещения прерывания, в ячейку памяти с адресом 0 заносится содержимое счетчика команд (адрес следующей команды основной программы, которая выполнялась бы в отсутствие прерывания), и управление передается команде, расположенной в ячейке 1.

Два последних действия имитируют выполнение команды "Обращение .к подпрограмме" (JSR 0), которая как бы вставлена между только что выполненной и следующей командами основной программы (см. параграф 2.4). Такую подпрограмму, текст которой должен написать сам пользователь, обычно называют подпрограммой обработки или обслуживания прерываний. Она всегда начинается в ячейке с адресом 1 (в ячейке 0 хранится адрес возврата из подпрограммы) и может иметь самый различный характер в зависимости от набора ВУ, подключенных к ЭВМ, и принципов построения основной программы. Чаще всего, однако, это подпрограмма требует использования аккумулятора, и поэтому одним из первых действий является запоминание содержимого аккумулятора и регистра переноса. Затем выясняется источник прерывания (путем опроса флагов определяется ВУ, флаг которого установлен в единичное состояние) и выполняется передача данных. Заканчивается подпрограмма восстановлением содержимого аккумулятора и регистра переноса, разрешением прерывания и выходом из подпрограммы на команду, адрес которой хранится в ячейке 0. Эти функции определяют содержание следующих шагов.

Шаг 3. Производится запоминание в памяти содержимого аккумулятора и регистра переноса. Для этого требуются минимум три команды: пересылка содержимого аккумулятора в специально отведенную буферную ячейку (MOV B1), циклический сдвиг (обычно влево) для того,

чтобы содержимое регистра переноса попало в аккумулятор (ROL), и запись этого содержимого в другую буферную ячейку (MOV B2). Таким образом, необходимый минимум информации о прерванной задаче оказывается обеспеченным – в ячейке 0 хранится адрес продолжения задачи, а в буферных ячейках запасены состояния двух других основных регистров машины в момент прерывания.

Шаг 4. Осуществляется поиск источника прерывания. Для этого в любой (наиболее целесообразной) последовательности опрашиваются флаги всех ВУ (команда TSF). При обнаружении ВУ с установленным флагом (флаг=1) выполняется переход к тому участку подпрограммы, в котором описаны действия по обмену информацией между процессором и этим внешним устройством.

Шаг 5. Выполняются передача данных и их предварительная обработка (если это необходимо).

Шаг 6. Восстанавливается содержимое регистра переноса и аккумулятора. Для этого требуются минимум пять команд: очистка аккумулятора (CLA), вызов содержимого буферной ячейки B2 в очищенный аккумулятор (ADD B2), циклический сдвиг вправо для восстановления содержимого регистра переноса (ROR), очистка аккумулятора (CLA) и вызов содержимого буферной ячейки B1 в очищенный аккумулятор (ADD B1).

Шаг 7. Контроллер прерываний вновь переводится в состояние разрешения, и осуществляется возврат к выполнению прерванной (основной) программы. Для этого последовательно выполняются команды EI ("Разрешение прерывания") и BR (0) - переход к команде с адресом, хранимым в ячейке 0 (см. параграф 2.4). Здесь следует отметить, что команда E1 должна обязательно располагаться непосредственно перед командой выхода из подпрограммы - BR (0). Это делается на случай очередного требования прерывания во время обработки рассматриваемого. Если между командами E1 и BR (0) поместить хотя бы одну команду, то после ее выполнения может произойти переход к обработке прерывания, содержимое ячейки 0 заменится новым адресом возврата, и таким образом путь возвращения к прерванной (основной) программе будет разрушен.

В заключение напомним следующее.

1. С помощью команды D1 контроллер прерываний устанавливается в состояние запрещения прерывания, и ЭВМ не реагирует на запросы обмена от любых внешних устройств. В то же состояние переходит этот контроллер и после нажатия кнопки ПУСК, поэтому для обеспечения обмена по прерыванию программы в последней обязательно должна присутствовать команда E1 ("Разрешение прерывания").

2. Прерывание программы наступает при наличии следующих условий: контроллер прерываний находится в состоянии разрешения прерывания, появился запрос на прерывание (флаг какого-либо ВУ

установился в единичное состояние) и окончилось выполнение любой текущей команды, кроме команд D1 ("Запрещение прерывания") и E1. Если в момент запроса на прерывание исполнялась команда E1, то состояние прерывания наступит после выполнения следующей за E1 командой (см. шаг 7).

3. В зависимости от конкретных условий подпрограмма обработки прерываний может иметь любой вид, но она всегда должна начинаться с команды, расположенной в ячейке с адресом 1, и заканчиваться командами E1 и BR (0), следующими друг за другом. Команда E1 служит для восстановления состояния разрешения прерывания, которое было автоматически заменено на запрещение прерывания при входе в подпрограмму, для того чтобы во время обработки текущего прерывания ЭВМ не реагировала на последующие.

Пример 3.3. Составить программу, которая периодически (с периодом в три цикла команды) наращивает на "единицу содержимое аккумулятора. Восемь младших разрядов последнего должны выводиться на ВУ-3 по запросу последнего (нажатие кнопки ВВОД N), а по запросу с ВУ-1 код, набранный на его тумблерах, должен помещаться в ячейку с адресом 25.

Основная программа решения такой задачи достаточно проста. Она может начаться с установки разрешения прерывания (E1) и очистки аккумулятора (CLA). Затем должен быть организован цикл для наращивания на единицу содержимого аккумулятора. Этот цикл может содержать всего две команды: "Инкремент аккумулятора" (INC) и команду безусловного перехода (BR) для возврата к INC. Однако по заданию такой цикл должен состоять из .трех команд, и поэтому между INC и BR можно поставить команду NOP ("Нет операции"). Подобная программа приведена в табл. 3.4.

Если команды этой программы занести в память ЭВМ, установить в счетчик команд пусковой адрес 0020 и нажать кнопку ПУСК, то начнет выполняться бесконечный цикл наращивания содержимого аккумулятора. В этом цикле все время будут повторяться команды: INC, NOP, BR 22, INC, NOP, BR 22, INC и т. д. Когда же на пульте управления (см. рис. 3.1) будет нажата любая из трех кнопок (ГОТОВ, ГОТОВ или ВВОД N), то произойдет передача управления команде, расположенной в ячейке 1, а в ячейку 0 будет записан адрес той из вышеперечисленных команд, которая выполнялась бы при отсутствии запроса на прерывание. Так осуществляется переход к подпрограмме обработке прерываний, которая должна быть расположена начиная с ячейки 1.

Таблица 3.4

Основная программа решения задачи примера 3.3

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
20	FA00	E1	Установка состояния разрешения прерывания
21	F200	CLA	Очистка аккумулятора
22	F200	INC	Цикл для наращивания содержимогоаккумулятора
23	F100	NOP	
24	C022	BR 22…	
25	0000		Ячейка для хранения кодов, поступающих из ВУ-1

В табл. 3.5 приведен один из возможных вариантов такой подпрограммы, построенной по стандартной схеме: запоминание содержимого основных регистров (шаг 3), отыскание источника прерывания (шаг 4), передача данных (шаг 5), восстановление содержимого основных регистров (шаг 6) и, наконец, организация выхода из подпрограммы (шаг 7). Подпрограмма предусматривает реакцию не только на запросы от ВУ-1 и ВУ-3, но и на запрос от ВУ-2, который может возникнуть от непреднамеренного нажатия на кнопку ГОТОВ около этого "устройства". В последнем случае производится очистка флага ВУ-2, что эквивалентно отмене непредусмотренного запроса.

Таблица 3.5

Первый вариант подпрограммы обработки прерываний для примера 3.3

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
0	0000		Ячейка для хранения адреса возврата (этот адрес будет занесен сюда на 2-м шаге)
1	C030	BR 30	Первая команда подпрограммы – переход к основному её тексту, расположенному в ячейках 30–4D
…	…	…	…
30	304C	MOV 4C	Сохранение в буферных ячейках 4C и 4D содержимого аккумулятора и регистра переноса **Шаг 3**
31	F600	ROL	
32	304D	MOV 4D	
33	E101	TSF 1	Опрос флага ВУ-1. Если он сброшен, то переход к опросу флага ВУ-2. В противном случае переход на ввод данных из ВУ-1 **Шаг 4**
34	C036	BR 36	
35	C039	BR 39	
36	E103	TSF 3	Опрос флага ВУ-3. Если он сброшен, то переход к опросу флага ВУ-2. В противном случае переход на ввод данных из ВУ-3
37	C043	DR 43	
38	C03E	BR 3E	
39	F200	CLA	Ввод данных из ВУ-1, пересылка их в ячейку 25, сброс флага ВУ-1, переход к восстановлению содержания основных регистров и выходу из подпрограммы **Шаг 5**
3A	E201	IN 1	
3B	E001	CLF 1	
3C	3025	MOV 25	
3D	C044	BR 44	
3E	F200	CLA	Пересылка в аккумулятор содержимого буферной ячейки 4C, вывод на ВУ-3 восьми младших разрядов аккумулятора, сброс флага ВУ-3, переход к восстановлению регистров и выходу
3F	404C	ADD 4C	
40	E303	OUT 3	
41	E003	CLF 3	
42	C044	BR 44	
43	E002	CLF 2	Очистка флага ВУ-2
44	F200	CLA	Восстановление содержимого регистра переноса и аккумулятора **Шаг 6**
45	404D	ADD 4D	
46	E700	ROR	
47	F200	CLA	
48	F400	CMA	
49	104C	AND 4C	
4A	FA00	E1	Возобновление состояния разрешения прерывания и выход из подпрограммы **Шаг 7**
4B	C800	BR (0)	
4C	0000		Ячейки для сохранения содержимого аккумулятора и регистра переноса
4D	0000		

Так как индексные ячейки 008-00F часто используются в циклических программах, то их нецелесообразно занимать под команды программы обработки прерываний. Поэтому в примере 3.3 основной текст такой подпрограммы размещен начиная с ячейки 30, а переход к ней осуществляется по команде BR 30, размещенной в ячейке 001.

Количество команд подпрограммы обработки прерываний может быть существенно сокращено, если учесть особенности решаемой задачи.

1. Накопленное в основной программе содержимое аккумулятора может быть испорчено (заменено на другое значение) лишь во время выполнения операции ввода данных с ВУ-1.

2. При любых (необходимых для решения задачи) взаимодействиях с ВУ-1, ВУ-2 или ВУ-3 содержимое регистра переноса не затрагивается.

3. На ВУ-3 должно выводится накопленное в основной программе содержимое аккумулятора, и оно не может быть испорчено во время этого вывода.

Вариант подпрограммы обработки прерываний, составленный с учетом перечисленных особенностей, приведен в табл. 3.6. Он короче первого на 10 команд (более чем 30%-ная экономия).

Таблица 3.6

Второй вариант подпрограммы обработки прерываний для примера 3.3

Адрес	Содержимое		Комментарии
	Код	Мнемоника	
0	0000		Адрес возврата из подпрограммы
1	0030	BR 30	Переход к выполнению подпрограммы
…	…	…	…
30	E101	TSF 1	Опрос флага ВУ-1 и переход к опросу флага ВУ-3,
31	C03C	BR 3C	если флаг ВУ-1 сброшен
32	303B	MOV 3B	Сохранение содержимого аккумулятора в ячейке 3B
33	F200	CLA	Ввод данных из ВУ-1 и пересылка их по заданному адресу (ячейка 25)
34	E201	IN 1	
35	E001	CLF 1	
36	3025	MOV 25	
37	F200	CLA	Восстановление содержимого аккумулятора
38	403B	ADD 3B	
39	FA00	E1	Восстановление состояния разрешения прерывания и выход из подпрограммы
3A	C800	BR (0)	
3B	0000		Буфер для сохранения содержимого аккумулятора
3C	E103	TSF 3	Опрос флага ВУ-3 и переход к сбросу флага ВУ-2,
3D	C041	BR 41	если флаг ВУ-3 сброшен
3E	E303	OUT 3	Вывод на ВУ-3 восьми младших разрядов аккумулятора и переход к завершающим командам подпрограммы
3F	E003	CLF 3	
40	C039	BR 39	
41	E002	CLF 2	Сброс флага ВУ-2 и переход к завершающим командам подпрограммы
42	C039	BR 39	

4. МИКРОПРОГРАММНОЕ УСТРОЙСТВО УПРАВЛЕНИЯ БАЗОВОЙ ЭВМ

4.1. Многоуровневые ЭВМ и их микропрограммный уровень

В настоящее время большинство пользователей ЭВМ начинает общение с этой машиной на одном из алгоритмических языков (СИ, ПАСКАЛе, БЕЙСИКе и т. п.), считая, что такой язык является языком машины. Вполне вероятно, что скоро будут конструировать подобные машины, но сейчас аппаратная реализация ЭВМ с алгоритмическими машинными языками весьма сложна и дорога. Поэтому машинные языки почти всех современных ЭВМ достаточно примитивны (похожи на язык базовой ЭВМ), что делает непосредственное использование таких языков неудобным и затруднительным. Возможность же исполнения на ЭВМ программы, написанной на алгоритмическом языке, обеспечивается с помощью специальных системных программ (компиляторов или интерпретаторов), осуществляющих перевод пользовательских программ на машинный язык ЭВМ.

Существуют два способа организации процесса перевода исходной программы с алгоритмического на машинный язык и процесса ее выполнения.

Первый способ, получивший название компиляции, заключается в том, что процесс выполнения алгоритма осуществляется лишь после завершения процесса перевода исходной программы. При этом в процессе выполнения полученной в результате перевода машинной программы уже не нужна не только исходная программа, но и программа-компилятор, осуществившая ее перевод в машинное представление.

Второй способ, получивший название интерпретации, заключается в том, что каждый оператор исходной программы заменяется программой-интерпретатором на эквивалентную последовательность машинных команд. Эта последовательность тут же выполняется, после чего переводится следующий оператор исходной программы, реализуются полученные при его переводе команды и т. д. Следовательно, во время решения задачи машине нужны и исходная программа, и программа-интерпретатор.

Затраты на создание компилятора (интерпретатора) и время на процесс перевода программы в значительной мере определяются сходством компилируемого и получаемого языков. Поэтому распространенные алгоритмические языки и более сложные языки программных систем не сразу переводят на язык машинных команд. Существует определенная иерархия языков программирования, в которой каждый более сложный язык базируется на предшествующем.

Мы уже сталкивались с одним из простейших языков, который очень близок к командам ЭВМ и обычно служит промежуточным языком

между ними и алгоритмическими языками. Это язык символического кодирования команд, часто называемый языком ассемблера. Языки ассемблеров (разработанные для каждого типа ЭВМ) - это первые средства автоматизации программирования в вычислительной технике. В них допускается использование символических имен и меток (адресов). Компиляторы с таких языков называются ассемблерами. Они отводят определенные ячейки памяти для символических переменных, организуют связи между различными частями программы, что резко облегчает программирование по сравнению с программированием на уровне команд.

Возможность реализации на любой ЭВМ программ, написанных на языках, отличающихся от ее машинного языка, лишний раз подчеркивает универсальность ЭВМ. Сейчас любой квалифицированный пользователь ЭВМ может создать свой собственный язык для описания каких-либо специфических задач (моделирования ЭВМ, управления прибором или ходом эксперимента, обучения игре в шахматы и т. п.) и написать для этого языка, например, интерпретатор на языке СИ или ПАСКАЛЬ

Человек, работающий с ЭВМ на том или другом языке (или обучающийся игре в шахматы), может не знать, выполняется ли его программа шаг за шагом интерпретатором, который, в свою очередь, выполняется другим интерпретатором, или же она осуществляется непосредственно аппаратными средствами (его обычно интересует лишь результат выполнения программы). Чаще всего ему кажется, что язык, на котором он общается с ЭВМ, – это ее машинный язык. Следовательно, разным пользователям одной и той же ЭВМ может казаться, что они работают на разных вычислительных машинах. Отсюда появились понятия: виртуальная (кажущаяся) ЭВМ и многоуровневая ЭВМ.

Многоуровневая ЭВМ – это вычислительная машина, имеющая средства для работы с n различными уровнями языков программирования (рис. 4.1).

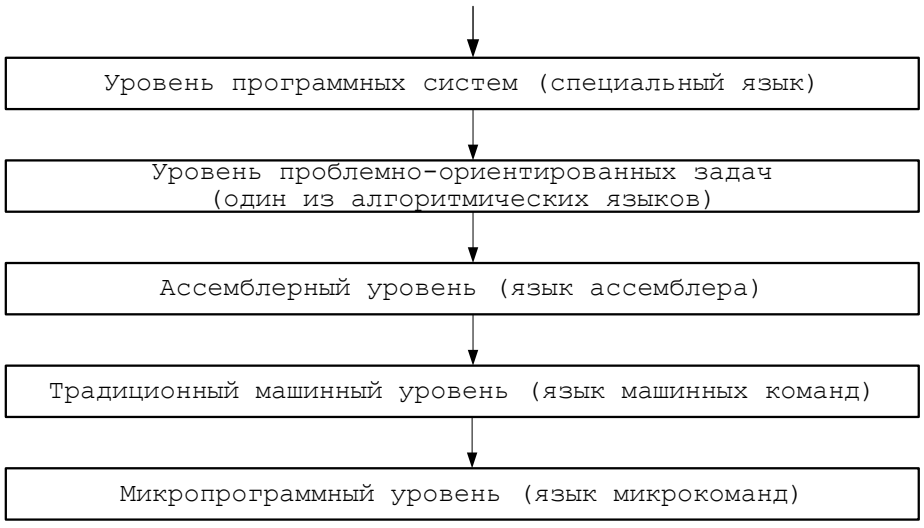


Рис 4.1. Многоуровневая ЭВМ

Нижний язык, или уровень, является наиболее простым, верхний – наиболее сложным. Такую машину можно рассматривать как n различных виртуальных машин, каждая из которых имеет свой машинный язык. Сложность аппаратурной реализации этих виртуальных машин возрастает по мере усложнения языка (увеличения номера уровня).

Если рассматривать с этих позиций базовую ЭВМ, то окажется, что язык команд, описанных в гл. 2, не является языком самого нижнего уровня. На ее нижнем уровне выполняются элементарные действия (микрооперации) над словами информации. Ряд из этих действий уже рассматривался при описании выполнения команд в параграфах 2.1 и 2.5 (пересылка содержимого одного регистра в другой регистр, проверка бита в каком-либо регистре и т. п.).

Управление порядком следования микроопераций осуществляется с помощью устройства управления базовой ЭВМ, которое, в свою очередь, является очень простой ЭВМ. Для этой ЭВМ регистры и вентильные схемы базовой ЭВМ служат как бы устройствами ввода и вывода (рис. 4.2). Программа работы такой ЭВМ – микропрограммного устройства управления (МПУ) – называется микропрограммой, а ее команды, содержащие информацию об элементарных действиях, подлежащих выполнению в течение одного рабочего такта ЭВМ, – микрокомандами.

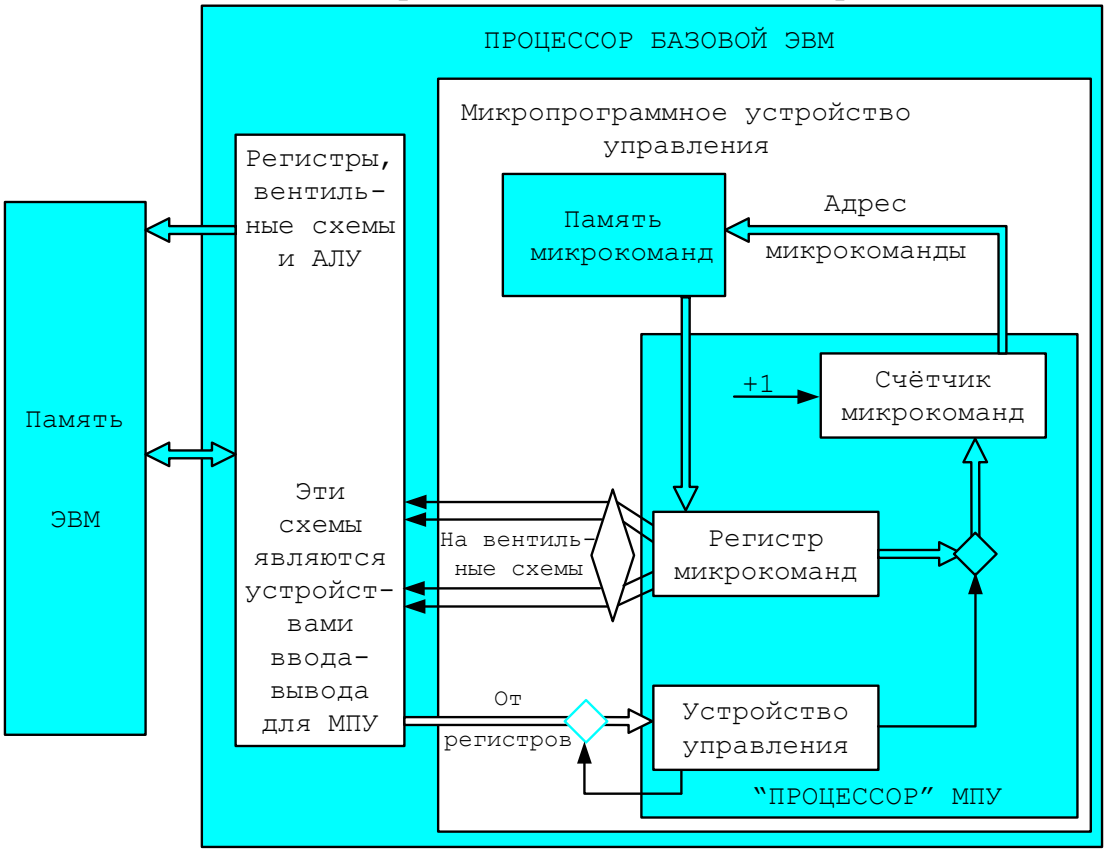


Рис 4.2. Структура микропрограммного устройства управления
и его связь с другими устройствами базовой ЭВМ

Микропрограмма хранится в постоянном запоминающем устройстве – памяти микрокоманд ..(МП). В каждом такте работы ЭВМ из МП в регистр микрокоманд пересылается очередная микрокоманда, т. е. микрокоманда, на которую указывает счетчик микрокоманд (он же регистр адреса микрокоманд). Затем счетчик микрокоманд наращивается на единицу.

Если из памяти выбрана так называемая операционная микрокоманда, биты которой определяют нужный набор микроопераций, то состояние этих битов передается на вентильные схемы процессора. Производятся соответствующая настройка АЛУ (на суммирование, логическое умножение, инвертирование и т. п.) и пересылка через него содержимого одних регистров в другие регистры (см. рис. 2.2, 2.8, 2.9 или 2.10).

При выборке управляющей микрокоманды в устройство управления МПУ пересылается содержимое указанного в микрокоманде регистра ЭВМ, из него выделяется указанный в микрокоманде бит и сравнивается с определенным битом микрокоманды (с 0 или 1). Если результат сравнения положителен, то в счетчик микрокоманд пересылается из микрокоманды адрес, по которому должна выбираться следующая микрокоманда микропрограммы. В противном случае никаких пересылок не производится и в следующем такте будет выполняться микрокоманда, расположенная вслед за исполняемой.

Следует ясно представлять, что у описываемой здесь микропрограммируемой ЭВМ имеются две памяти, два набора команд и две программы: традиционного машинного уровня (команды табл. 2.4) и микропрограммного уровня. Однако аппаратно реализованный процессор - только один, а архитектура машины формирует микропрограммный уровень.

Традиционный машинный уровень - это один из тех машинных уровней, описание которого можно найти в руководстве по эксплуатации ЭВМ. В рассматриваемом случае базовая ЭВМ рекламировалась бы как ЭВМ, обладающая памятью емкостью 2048 16-битовых слов и системой команд, состоящей из 28 команд. Среди этих команд: логическое умножение, сложение, вычитание, условные и безусловный переходы, обращение к подпрограмме и т. п. Она имеет два вида адресации (прямую и косвенную), к ней можно подключать до 256 внешних устройств и т. д. В руководстве можно найти упоминание о том, что машина микропрограммируема, однако описания микропрограммного уровня, памяти микрокоманд и самих микрокоманд там обычно не приводится.

Пользователю, желающему программировать на традиционном машинном уровне (и тем более на языках высокого уровня), в конечном счете безразлично, реализован ли традиционный уровень путем создания специальных электрических схем или же для этих целей используется

микропрограмма, выполняемая на более низком машинном уровне. Однако в связи с появлением ряда микропроцессорных наборов, имеющих единственный язык - язык микрокоманд, возникла необходимость в ознакомлении пользователей с программированием на уровне микрокоманд.

4.2. Компоненты процессора и основные операции

В параграфах 1.2 и 2.1 уже давалось описание большинства компонентов процессора: регистров, шин, вентильных схем и генератора тактовых импульсов. Здесь подробнее рассмотрим способы передачи данных между регистрами, работу АЛУ и ряда вспомогательных схем.

Передача данных между регистрами. Это основная операция микропрограммного уровня. Можно указать четыре типа такой передачи данных: прямая, асимметричная, по частям и групповая (рис. 4.3).

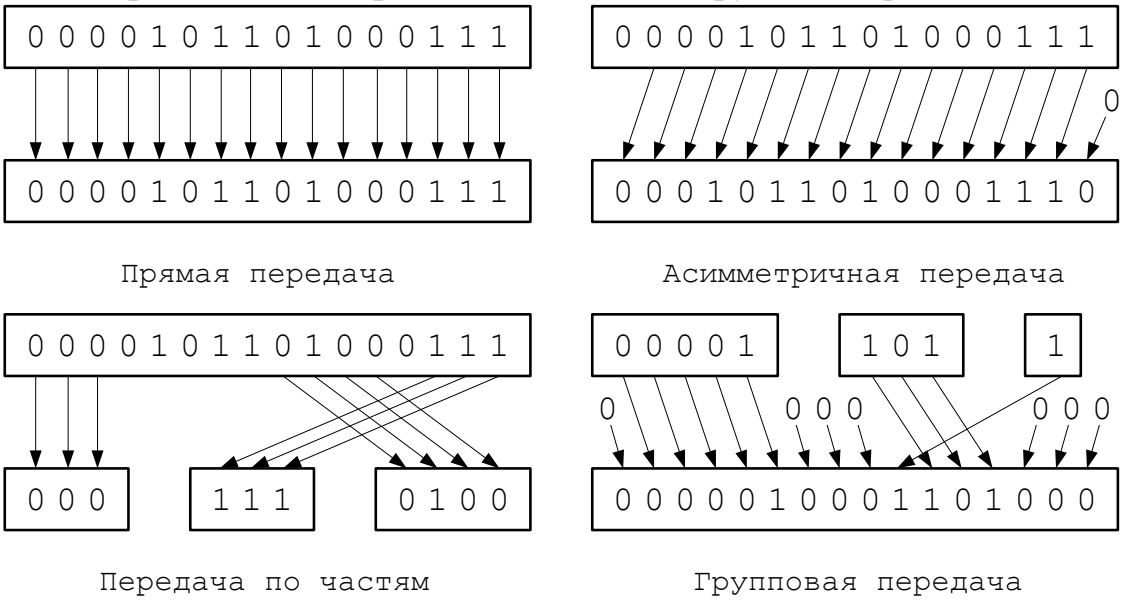


Рис. 4.3. Передача данных между регистрами

В случае прямой передачи содержимое N-битового регистра копируется в N-битовом регистре так, что i-й бит копии соответствует i-му биту оригинала. При асимметричной передаче i-й бит копии соответствует (i+K)-му биту оригинала, где K - положительное или отрицательное целое число. В этом случае для одного или нескольких разрядов регистра-приемника может не оказаться соответствующих битов из регистра-источника. Такие разряды заполняются нулями.

При передаче по частям содержимое одного регистра копируется (оказывается "разбросанным") в нескольких регистрах, причем порядок расположения битов может быть изменен. В случае групповой передачи в различные части регистра-приемника записываются копии содержимого нескольких регистров, порядок расположения битов также может быть

изменен. Разряды, не имеющие соответствующих источников информации, заполняются нулями.

Передача данных между регистрами изменяет содержимое регистра-приемника, но сохраняет неизменным содержимое регистра источника. Для передачи используются шины различной ширины с встроенными в них вентильными схемами (рис. 1.3).

На рис. 4.4 показана работа устройства сдвига.

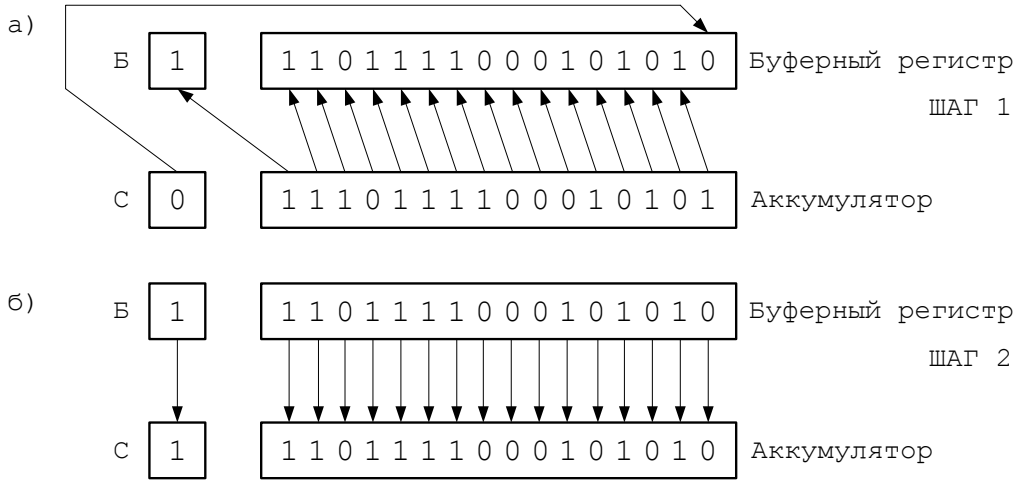


Рис. 4.4. Работа устройства сдвига на 1 бит влево

Число, подлежащее циклическому сдвигу на 1 бит влево, помещается в аккумулятор. Затем выполняется асимметричная передача во внутренние регистры АЛУ: буферный регистр и однобитовый регистр Б. При этом младший бит буферного регистра заполняется не нулем, а содержимым регистра переноса С. После этого производится прямая передача из Б и буферного регистра в С и аккумулятор.

Следует заметить, что обе эти передачи осуществлялись с помощью различных шин. Одна из них (с асимметричной связью разрядов регистра-источника с регистром-приемником) служит для передачи информации из аккумулятора в буферный регистр, а другая - из буферного регистра (БР) в аккумулятор. Для того чтобы данная пара регистров могла осуществлять и сдвиг вправо, необходимо использовать еще один информационный канал (шину с вентильной схемой) с правой асимметрией.

Работа арифметико-логического устройства. АЛУ использует содержимое аккумулятора и регистра данных в качестве операндов для получения результата, который помещается в аккумулятор. Все арифметические и логические команды базовой ЭВМ (см. табл. 2.4) и вспомогательные арифметические операции (например, увеличение на единицу содержимого счетчика команд) можно выполнить с помощью АЛУ, структурная схема которого приведена на рис. 4.5. Кратко рассмотрим схемную реализацию отдельных микроопераций, выполняемых АЛУ, аккумулятором и регистром переноса (С) по управляющим сигналам У7-У17 и У22, поступающим на вентильные

схемы В7-В17 и В22 (на рис. 4.5 управляющие сигналы У7-У17 и У22 не показаны).

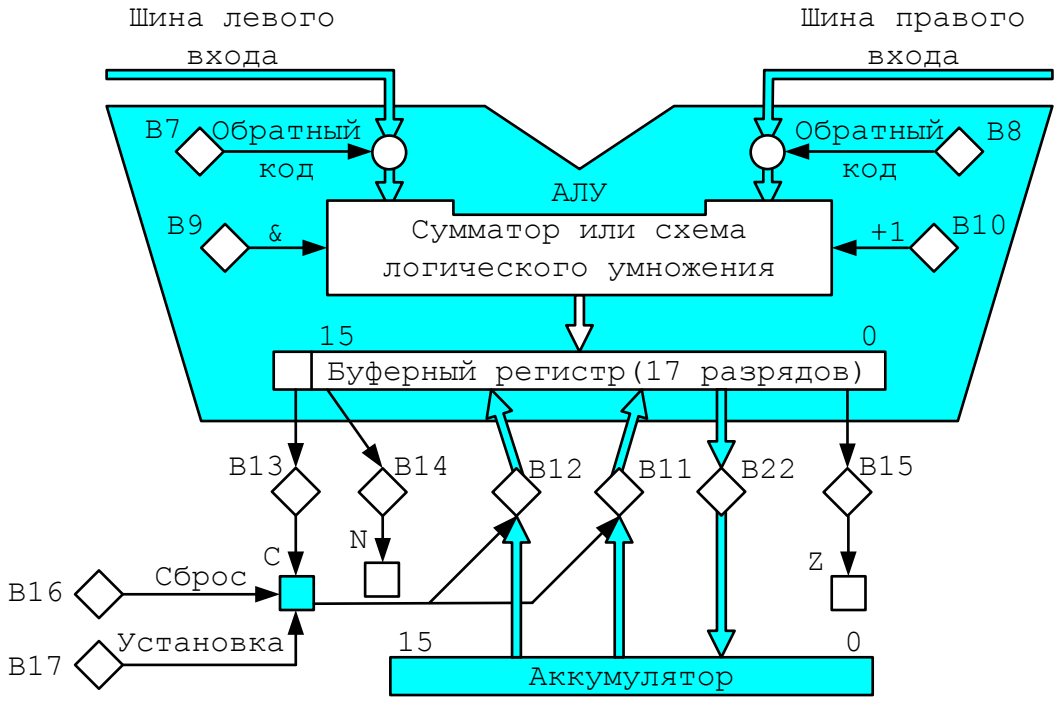


Рис. 4.5. АЛУ, аккумулятор и регистр переноса

Обратный код входных сигналов АЛУ может быть получен по схеме рис. 4.6, а), в которой изменение всех 0 на 1 и всех 1 на 0 осуществляется с помощью инверторов СхНЕ. При отсутствии управляющего сигнала (см. рис. 1.3) единица не проходит на выход В7, вентиль ВИ2 закрыт, на выходе инвертора НЕ единичный сигнал и вентиль ВИ1 открыт. В этом случае в узле Вых.- прямой код операнда, т. е. код, совпадающий с кодом в узле Вх. Когда на В7 подается единичный управляющий сигнал, через него проходит *1*, создаваемая соответствующим постоянным напряжением (см. рис. 1.2), закрывается ВИ1, открывается ВИ2 и в узел Вых. подается код, обратный по отношению к коду в узле Вх.

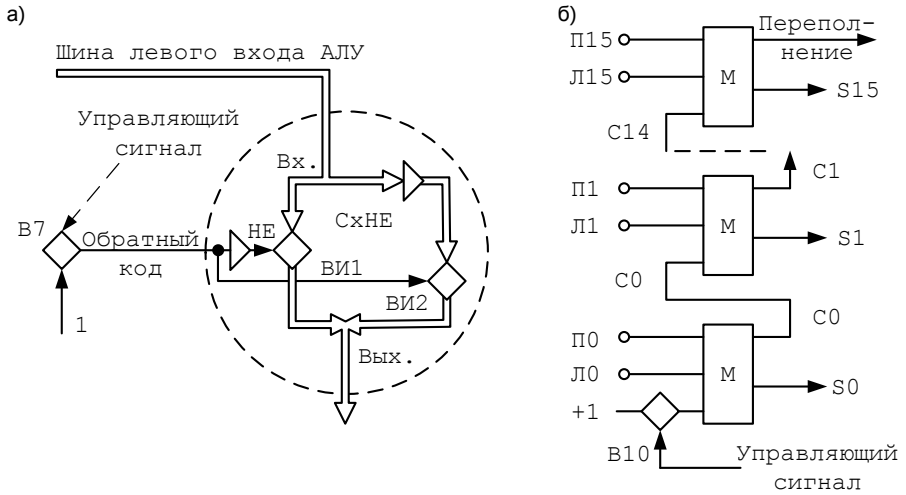


Рис. 4.6. Схемы получения обратного кода (а) и суммы (б)

Сложение осуществляется с помощью типовой схемы сумматора (рис. 4.6, б). Функция, реализуемая одним из разрядов этого сумматора (разрядом с номером 1), представлена в виде табл. 4.1. Входными переменными таблицы являются С0, Л1 и П1, выходными переменными – S1 и С1. При закрытом вентиле В10 сумматор реализует зависимость S = Л + П.

Таблица 4.1

Таблица выходов для одного разряда двоичного сумматора SM

Перенос из предыдущего разряда С0	Слагаемые		Сумма	Перенос в следующий разряд С1
	Левый вход Л1	Правый вход П1		
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

С помощью управляющего сигнала, подаваемого на вентиль В10, можно обеспечить прибавление к сумме единицы: S = Л + П + 1.

Перенос из старшего разряда (разряда с номером 15) сигнализирует о переполнении сумматора. Этот перенос записывается в старший разряд, (разряд с номером 16) буферного регистра (см. рис. 4.5) и может быть переписан в регистр переноса С, путем подачи управляющего сигнала на вентиль В13.

Следует отметить, что сумматор, вентильная схема и инвертор – это так называемые комбинационные схемы, т. е. схемы, в которых значения выходных переменных полностью определяются значениями входных переменных в данный момент времени. Следовательно, изменение любого из входных сигналов АЛУ сразу же приведет к изменению суммы. Это позволяет выполнять с помощью сумматора и схем получения обратного кода несколько различных микроопераций при различных сочетаниях управляющих сигналов, подаваемых на вентильные схемы Bl–B8 и В10 (рис. 4.5 и 4.7):

1) суммирование содержимого двух регистров, один из которых подключен к левому входу АЛУ, а – другой – к правому.

Для этого надо подать единичный управляющий сигнал на В/ или В2, или В3 и такой же сигнал на В4 или В5, или В6 (при выполнении команды ADD управляющие сигналы подаются на В1 и В4);

2) суммирование содержимого двух регистров и 1 (три слагаемых).

Эта микрооперация отличается от предыдущей лишь тем, что одновременно с подачей управляющих сигналов на вентили, устанавливающие связь между сумматором и регистрами (где хранятся коды слагаемых), такой же сигнал должен подаваться на вентиль B10;

3) вычитание.

Например, при вычитании содержимого регистра данных из содержимого аккумулятора (команда SUB в табл. 2.4) надо одновременно подать единичные управляющие сигналы на B1, B4, B8 и B10. Тогда к уменьшаемому прибавится обратный код вычитаемого и к этой сумме добавится единица, что эквивалентно суммированию уменьшаемого с дополнительным кодом вычитаемого (см. параграф 2.3);

4) добавление единицы к содержимому какого-либо регистра.

Эта микрооперация нужна для наращивания содержимого счетчика команд, выполнения команды INC (см. табл. 2.4) и в ряде других случаев. Для ее реализации необходимо подать управляющие сигналы на B10 и вентильную схему, устанавливающую связь между сумматором и соответствующим регистром. При этом другой вход сумматора будет соединен с шиной, все вентильные схемы которой закрыты, что эквивалентно пересылке по шине кода числа 0;

5) вычитание единицы из содержимого какого-либо регистра.

Эта микрооперация нужна, например, для реализации команды DEC (см. табл. 2.4). Управляющие сигналы подаются на вентильные схемы B4 и BS, т. е. содержимое аккумулятора суммируется с обратным кодом числа 0 или (что то же самое) с дополнительным кодом числа – 1 (см. параграф 2.3);

6) инвертирование содержимого какого-либо регистра.

Такое преобразование уже использовалось в микрооперациях пп. 3 и 5. Для выполнения же, например, команды CMA (см. табл. 2.4) следует подать управляющие сигналы на B4 и B7;

7) очистка какого-либо регистра (установка в нем кода числа 0).

Для этого надо выполнить две микрооперации: засылку нуля в БР (в данном такте закрыты все вентили с Bl по B10) и пересылку содержимого БР в нужный регистр (открыт соответствующий вентиль на выходной шине БР, например B22 при выполнении команды CLA).

Список подобных микроопераций можно было бы продолжить (например, занесение в регистр кода числа – 1, получение сигнала переноса из старшего разряда сумматора при сложении двух чисел, равных –1, и т. д.), однако и приведенного перечня достаточно для подтверждения следующего положения:

чтобы интерпретируемая машина выполняла программу, необходимо в определенной последовательности открывать и закрывать вентильные схемы; описание того, какую вентильную схему и когда открывать,

составляет программу (микропрограмму) для машины, система команд которой включает команду "Открыть вентильную схему".

Логическое умножение осуществляется с помощью вентильной схемы, на входы которой подаются коды операндов, а на выходе образуется искомый результат (см. параграфы 1.2 и 2.5).

В АЛУ сумматор и схема логического умножения объединены в один блок. В блоке входные шины разветвляются на оба эти устройства, а выходы устройств через вентильные схемы подключены ко входам буферного регистра. Следовательно, при поступлении операндов одновременно выполняются операции суммирования и логического умножения, но так как по сигналу с В9 (см. рис. 4.5 и 4.7) открыта лишь одна из выходных вентильных схем, то на буферный регистр поступит только один из двух результатов. Управление выходными вентильными схемами осуществляется подобно тому, как управляются ВИ1 и ВИ2 на рис. 4.6: при отсутствии управляющего сигнала на В9 в БР записывается сумма, а при наличии управляющего сигнала на В9 - результат логического умножения.

Циклический сдвиг на один разряд вправо или влево производится путем подачи единичного управляющего сигнала на вентильную схему~ Bl 1 или В12 соответственно, а затем на вентильные схемы В13 и В22 (см. рис. 4.4 и 4.5).

Установка признаков переноса из старшего разряда сумматора, а также отрицательного или нулевого значения результата осуществляется с помощью посылки управляющих сигналов на вентили В13, В14 и В15 соответственно.

При подаче управляющего сигнала на вентиль В14 выполняется перепись содержимого 15-го (знакового) разряда БР в однобитовый регистр N. При открывании вентиля В15 производится перепись в однобитовый регистр Z содержимого специальной схемы, выходной сигнал которой равен единице только тогда, когда во всех 16 разрядах буферного регистра (с 0-го по 15-й) содержатся нули, т. е. когда в БР хранится код числа 0.

Содержание регистра переноса также может быть изменено с помощью управляющих сигналов. Подача сигнала на В13 позволяет переписать в С содержимое старшего разряда БР (разряда с номером 16), в котором хранится перенос из старшего разряда сумматора. Посылка управляющего

сигнала на B16 приводит к сбросу C (команда CLC - очистка регистра переноса), а подача сигнала на B17 – к записи в него единицы.

Регистр состояний. Это объединение однобитовых регистров признаков и состояний, назначение которых было рассмотрено здесь и в параграфе 3.3, сделано в целях формального уменьшения числа регистров, с которыми работает микропрограммное устройство управления, что позволяет сократить разрядность управляющих микрокоманд.

На втором отладочном пульте базовой ЭВМ (см. рис. 4.7) регистр состояний (PC) изображён (для удобства работы) в виде самостоятельного регистра. На самом деле его состояния лишь дублируют состояния однобитовых регистров C, Z, N и т. д. Соответствие между этими регистрами и разрядами PC приведено в табл. 4.2.

Таблица 4.2

Регистр состояния

Разряд	Содержимое
0	Перенос (C)
1	Нуль (Z)
2	Знак (N)
3	0 - используется для организации безусловных переходов в МПУ
4	Разрешение прерывания
5	Прерывание
6	Состояния ВУ (Ф)
7	Состояние тумблера РАБОТА/ОСТАНОВ (1 - РАБОТА)
8	Программа
9	Выборка команды
10	Выборка адреса
11	Исполнение
12	Ввод-вывод

Вентильные схемы. Ниже перечислены функции ряда вентильных схем, используемых для пересылки информации между регистрами, между регистрами и АЛУ и для управления микрооперациями в АЛУ. Функции остальных вентильных схем на рис. 4.7 будут рассмотрены в параграфе 4.3.

Отметим, что обмен данными между регистрами, имеющими разную длину (например, между БР и СК), выполняется с помощью прямой

передачи (см. рис. 4.3), и старшие разряды более длинного приемника заполняются нулями.

Входные сигналы АЛУ. Вентильные схемы В1, В2 и В3 предназначены для правого входа АЛУ, а В4, В5 и В6 - для левого. Все они используются для пересылки содержимого соответствующих регистров в буферный регистр АЛУ и далее в микропрограммное устройство управления.

В1. РД в АЛУ. Используется при арифметических и логической операциях, а также при передачах через АЛУ в РК, СК, РА и МПУ.

В2. РК в АЛУ. Используется для передачи содержимого РК в МПУ.

В3. СК в АЛУ. Используется для увеличения на 1 содержимого СК и передачи через АЛУ в РА, РД и МПУ.

В4. А в АЛУ. Используется при арифметических и логической операциях, а также при передачах через АЛУ в РД и МПУ.

В5. РС в АЛУ. Используется для передачи содержимого РС в МПУ.

В6. КР в АЛУ. Используется во время работы с пультом управления ЭВМ для передачи содержимого клавишного регистра (КР) в СК и РД.

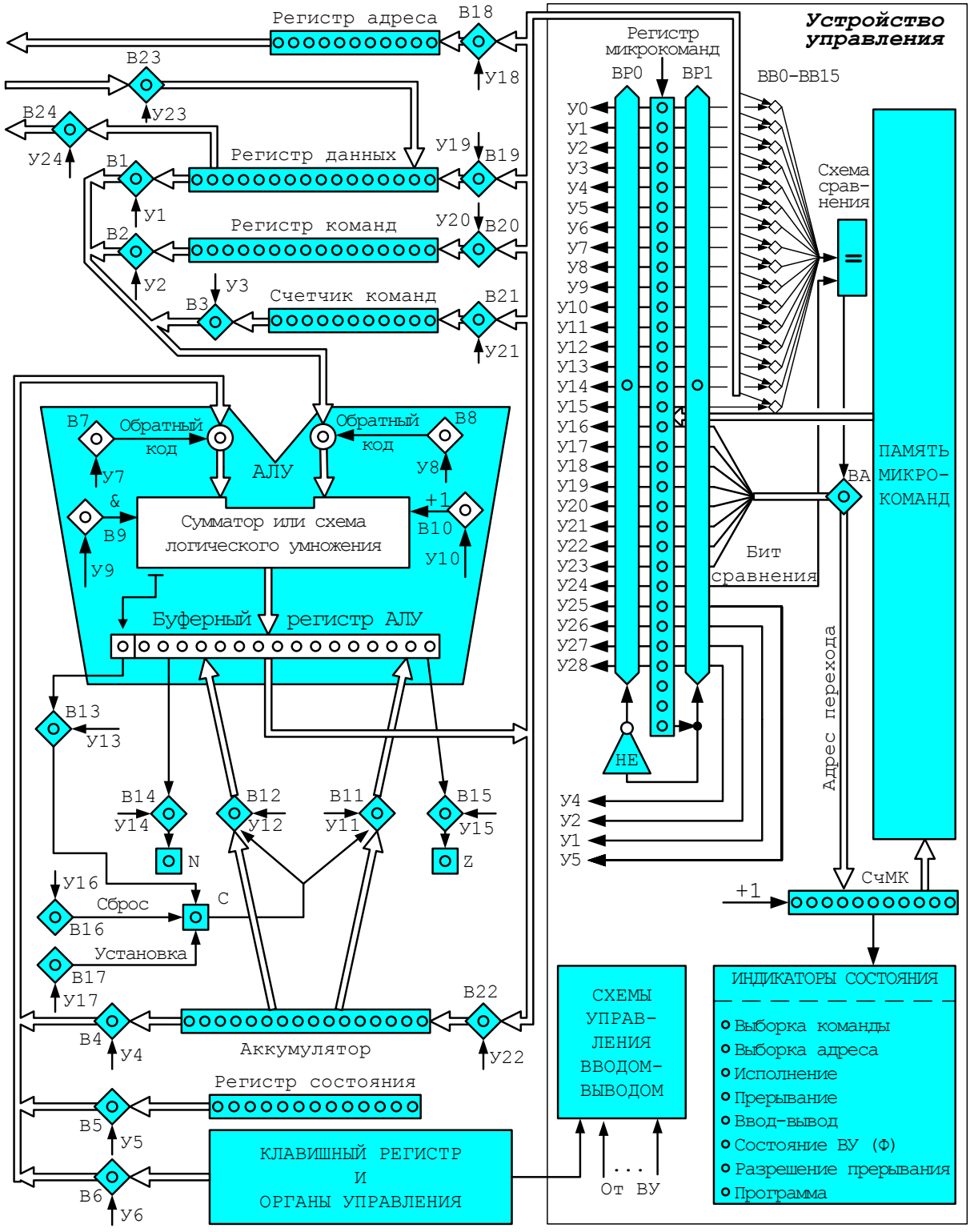


Рис. 4.7. Второй отладочный пульт базовой ЭВМ

Арифметические операции и сдвиги.

В7. Когда эта вентильная схема открыта, сигнал на левый вход сумматора поступает в обратном коде.

B8. Когда эта вентильная схема открыта, сигнал на правый вход сумматора поступает в обратном коде.

B9. Когда эта вентильная схема открыта, в БР поступает результат логического умножения операндов. При закрытой схеме в БР поступает сумма операндов.

B10. Когда эта вентильная схема открыта, к сумме операндов добавляется 1.

B11. Когда эта вентильная схема открыта, содержимое А и С асимметрично пересылается в БР: содержимое младшего разряда А попадает в
16-й бит БР, содержимое С - в 15-й бит БР, содержимое старшего разряда А - в 14-й бит БР и т. д.

B12. Когда эта вентильная схема открыта, содержимое А и С асимметрич-но пересылается в БР: содержимое младшего разряда А попадает в 1-й бит БР, содержимое С - в 0-й бит БР, содержимое старшего разряда А – в 16-й бит БР и т. д.

Установка признаков.

B13. Когда эта вентильная схема открыта, перенос (0 или 1) из старшего разряда сумматора, зафиксированный в старшем разряде БР, поступает в С.

B14. Когда эта вентильная схема открыта, знак числа, хранящегося в БР, поступает в N.

B15. Когда эта вентильная схема открыта, в Z поступает 1, если содержи-мое БР равно 0, и 0, если содержимое БР не равно 0.

B16. Когда эта вентильная схема открыта, в С записывается 0.

B17. Когда эта вентильная схема открыта, в С записывается 1.

Выходной сигнал АЛУ (БР).

B18. БР в РА. Используется при передаче в РА содержимого РД или СК.

B19. БР в РД. Используется при передаче в РД содержимого СК, А или КР.

B20. БР в РК. Используется при передаче в РК содержимого РД.

B21. БР в СК. Используется при передаче в СК содержимого РД или КР и для наращивания СК на 1.

B22. БР в А. Используется для пересылки результатов арифметических операций и сдвигов в А.

Чтение из памяти интерпретируемой машины и запись в эту память.

B23. Из памяти в РД. Используется для загрузки в РД слова памяти, адрес которого указан в РА.

B24. Из РД в память. Используется для записи содержимого РД в слово памяти, адрес которого указан в РА.

Организация ввода-вывода информации.

B25. Когда эта вентильная схема открыта, в контроллеры ВУ из РД пересылается приказ на ввод-вывод и адрес требуемого ВУ.

B26. Когда эта вентильная схема открыта, на все контроллеры ВУ одно-временно поступает сигнал, сбрасывающий их флаги.

B27. Когда эта вентильная схема открыта, производится сброс бита разрешения прерывания в РС (бит 4).

B28. Когда эта вентильная схема открыта, производится установка бита разрешения прерывания в РС (бит 4).

Останов ЭВМ.

B0. Эта вентильная схема используется для передачи сигнала прекращения выполнения программы (команда HLT).

Другие компоненты процессора (память микрокоманд, регистр микроко-манд и т. д.) рассмотрены в параграфе 4.3.

Нетрудно заметить, что в предложенной структуре передача данных между регистрами осуществляется только через буферный регистр АЛУ. Это сделано для уменьшения числа шин и вентилей, т. е. для упрощения и удешев-ления процессора. Однако такое решение замедляет процедуру обмена.

4.3. Микропрограммное управление вентильными схемами

Микрокоманды. Интерпретирующий уровень (см. рис. 4.7) обладает на-бором команд, состоящим только из двух микрокоманд: операционной (ОМК) и управляющей (УМК), форматы которых показаны на рис. 4.8. Поскольку команд только две, достаточно однобитового кода операции: 0 – для ОМК и 1 – для УМК. Хотя для размещения информации и кода операции достаточно 30 бит, были выбраны 32-разрядные (четырехбайтовые) микрокоманды.

В операционной микрокоманде каждый из 29 информационных битов управляет одной из 29 вентильных схем, описанных в параграфе 4.2. Единица означает, что вентильная схема должна быть открыта, нуль -

закрыта. Подобная структура микрокоманды обычно называется горизонтальной.

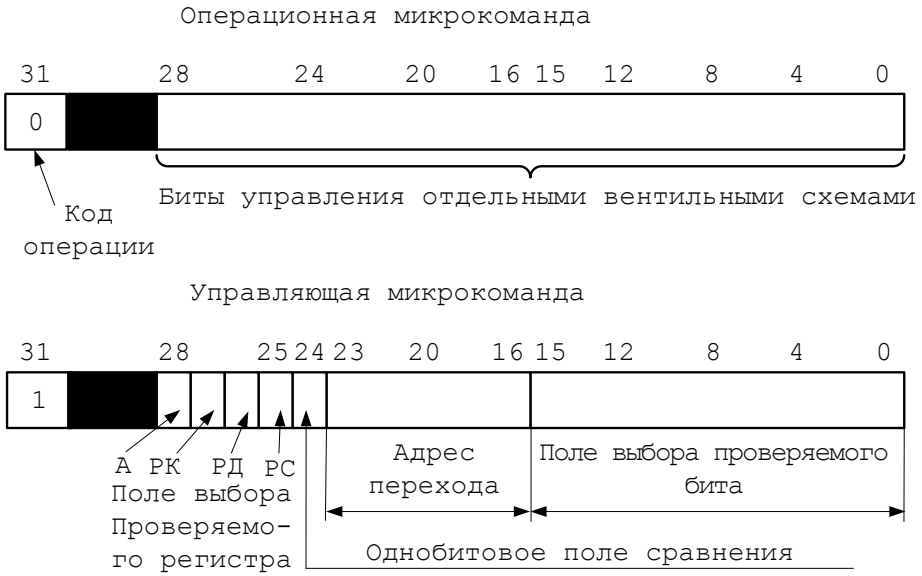


Рис. 4.8. Система команд микропрограммного уровня, показанного на рис. 4.7 (состоит только из двух команд)

Управляющая микрокоманда используется для изменения последовательности выполнения микрокоманд в зависимости от тех или иных условий. Каждая микрокоманда УМК определяет 1 бит, подлежащий проверке (любой из 61 бит, находящихся в А, РК, РД и РС). Если выбранный бит и 24-й бит микрокоманды совпадают, последовательность микрокоманд, подлежащих исполнению, изменяется, и адрес следующей микрокоманды определяется битами с 16 по 23 самой УМК. Если выбранный бит отличается от бита 24, последовательность выполнения микрокоманд не претерпевает изменений.

Бит, подлежащий проверке, определяется двумя полями микрокоманды. Биты с 25 по 28 указывают, какой регистр интерпретирующей машины должен быть проверен согласно следующим правилам:

бит 25 - регистр состояний (РС);

бит 26 - регистр данных (РД);

бит 27 - регистр команд (РК);

бит 28 - аккумулятор (А).

Биты микрокоманды с 0 по 15 указывают, какой бит выбранного регистра подлежит проверке. Только один бит, принадлежащий группе битов с 0 по 15, и один бит группы битов с 25 по 28 должны быть равны единице.

Например, после увеличения на единицу содержимого регистра данных (РД) в команде ISZ (см. параграф 2.4) надо проверить его знаковый разряд (бит с номером 15). Если этот разряд равен 1 (содержимое РД отрицательное), то выполнение микрокоманд команды ISZ завершается. В противном случае необходимо прибавить единицу к содержимому счетчика команд, т. е. организовать пропуск следующей за ISZ команды. Разветвление в микропрограмме реализации команды ISZ (переход по адресу 8F) осуществляется с помощью микрокоманды $(858F8000)_{16}$. На рис. 4.9,а эта микрокоманда представлена в двоичной форме с указанием всех используемых битов.

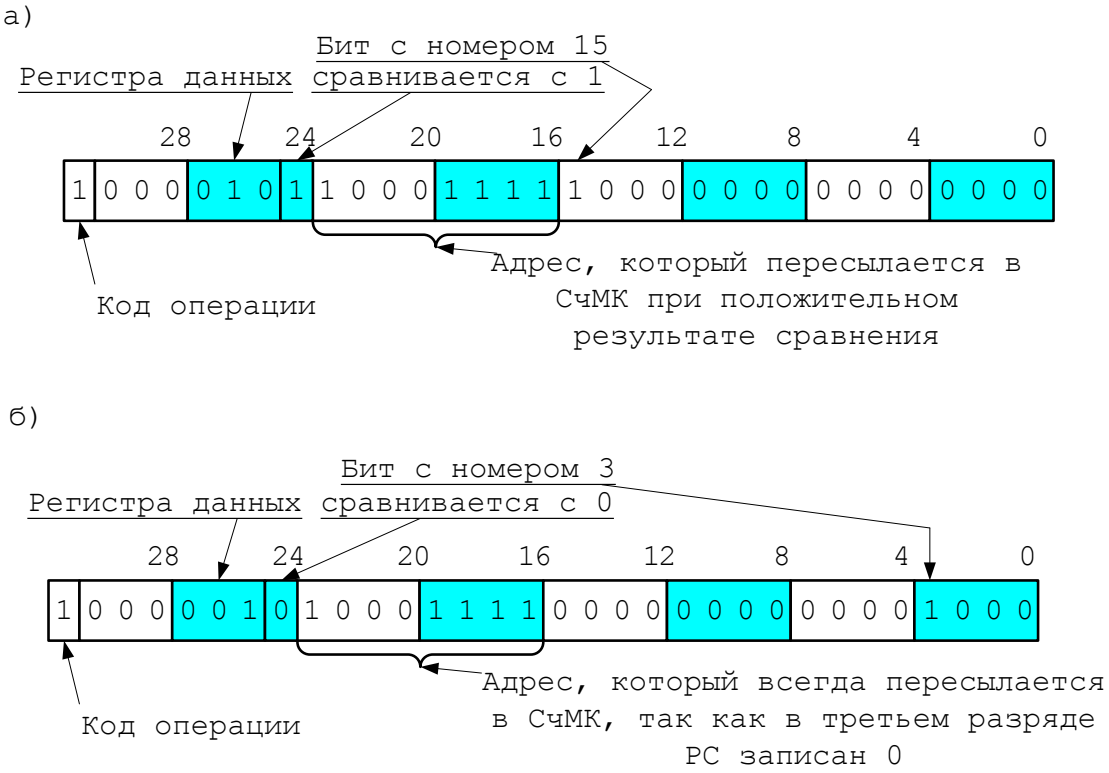


Рис. 4.9. Примеры управляющих микрокоманд:
а - условного перехода; б - безусловного перехода

Для организации безусловного перехода (например, по тому же адресу 8F) используется 3-й бит регистра состояний, содержащий константу 0. Сравнение этого разряда с константой 0, установленной в 24-й разряд УМК, всегда дает положительный результат и позволяет переслать в СчМК нужный адрес перехода (рис. 4.9,б).

Выполнение микропрограмм. Микропрограмма хранится в памяти микрокоманд, которая состоит из 256 (2^8) 32-битовых слов (см. рис. 4.7). Для управления выполнением микропрограммы используется счетчик микрокоманд (СчМК), аналогичный счетчику команд традиционного машинного уровня. Чтобы извлекать микрокоманды из памяти

микрокоманд, декодировать их и исполнять, следовало бы иметь регистры адреса, данных и команд. Но поскольку машина-интерпретатор очень проста, один регистр выполнит функции регистра адреса и счетчика микрокоманд, а другой – регистра данных и регистра микрокоманд (РМК).

Работу процессора можно описать следующим образом. При появлении тактового импульса в РМК из памяти микрокоманд извлекается и загружается слово, на которое указывает СчМК, и к содержимому этого счетчика прибавляется единица. Если из памяти извлечена УМК, то в 31-м бите РМК содержится 1 (код операции УМК), которая открывает вентильную схему ВР1 и тем самым создает условия для исполнения УМК. Если же извлечена ОМК, то в 31-м бите РМК – нуль. Этот сигнал с помощью инвертора НЕ открывает вентильную схему ВР0, и через нее на В0-В28 передаются состояния соответствующих битов РМК. Разряды РМК, содержащие единицы, создают открывающий управляющий сигнал, а содержащие нули - закрывающий (У0 – У28).

При исполнении УМК по сигналу, создаваемому каким-либо битом поля выбора проверяемого регистра (У1, У2, У4 или У5), открывается одна из вентильных схем В1, В2, В4 или В5 и на вентили ВВ0 – ВВ15 поступает через АЛУ содержимое соответствующего регистра. Одновременно на эти же вентили поступает с РМК содержимое поля выбора проверяемого бита. Так как в этом поле должна быть записана только одна единица, то открывается лишь один из вентилей ВВ0 - ВВ15, через который на схему сравнения поступает содержимое проверяемого бита из проверяемого регистра.

На второй вход схемы сравнения поступает содержимое однобитового поля сравнения (24-й бит УМК), в которое при кодировании УМК записали цифру 0 или 1. Если проверяемый бит и бит из поля сравнения идентичны, то схема сравнения формирует единичный сигнал, который открывает вентильную схему ВА, и в СчМК пересылается адрес перехода (биты с 16 по 23). В противном случае в СчМК сохраняется адрес микрокоманды, расположенной вслед за исполняемой.

Когда в СчМК появляется адрес микрокоманды, с которой начинается какой-либо из машинных циклов (см. параграфы 2.5 и 3.3), изменяется информация на индикаторе состояний процессора. Эта информация изменяется и при переходе ЭВМ из состояния ОСТАНОВ в состояние ПРОГРАММА или обратном переходе.

4.4. Интерпретатор базовой ЭВМ

Язык микропрограммирования. Микропрограммы можно писать в виде последовательности 32-битовых двоичных чисел, однако такая работа весьма утомительна, и можно легко допустить ошибку. Для облегчения записи и проверки микропрограмм выполнения отдельных машинных команд и циклов используем их шестнадцатеричное кодирование. Комментарии к микро-командам будем записывать с помощью мнемонического языка микро-программирования, напоминающего алгоритмические языки высокого уровня (СИ, ПАСКАЛЬ и др.). Рассмотрим примеры такой записи для различных микроопераций.

Пересылка информации из регистра в регистр (регистры) комментируется посредством оператора присваивания. Например, пересылка содержимого БР в РД (открыть B19) записывается в виде

0008 0000; БР => РД.

(Для облегчения шестнадцатеричного кодирования используйте рис. 4.9, где представлено разбиение 32-битового регистра на тетрады.)

Пересылка содержимого БР в аккумулятор с установкой признаков результата (открыть B13, B14, B15 и B22) записывается в виде

0040 E000; БР => A, C, N, Z.

Обмен информацией между памятью (ОП) и регистром данных задается следующим образом:

чтение из памяти по адресу, хранимому в РА (открыть B23),

0080 0000; ОП (РА) => РД;

запись в память по адресу, хранимому в РА (открыть B24),

0100 0000; РД => ОП (РА).

Операции, выполняемые в АЛУ, также комментируются операторами присваивания. При этом обратный код записывается как функция COM, а сдвиги вправо и влево - как функции RAR и RAL соответственно. Например, суммирование А и РД (открыть B1 и B4) запишется в виде

0000 0012; А + РД => БР,

вычитание РД из А (открыть B1, B4, B8 и B10)

00000512; А + COM (РД) + 1 => БР,

циклический сдвиг содержимого А и С на один разряд вправо (открыть Bl1)

0000 0800; RAR (A, C) => БР

и, наконец, прибавление единицы к содержимому СК (открыть B3 и B10)

0000 0408; СК + 1 => БР.

Управляющая микрокоманда комментируется с помощью оператора условного перехода

IF BIT (к, рег) = В THEN метка (адрес),

где к – номер бита, подлежащего проверке (от 0 до 15); рег – регистр, подлежащий проверке (А, РК, РД, РС); В - 0 или 1; метка (адрес) - метка и адрес микрокоманды, которой передается управление, если проверка дает положительный результат.

Например, проверка бита 12 регистра команд запишется так:

89xx 1000; IF BIT (12, РК) = 1 THEN El (xx),

где вместо "xx" должен быть проставлен адрес микрокоманды, которой передается управление при положительном результате проверки; комментарии этой микрокоманды должны иметь метку E1.

Проверка 15-го бита аккумулятора запишется в виде

90xx 8000; IF BIT (15, А) = 0 THEN НЕ (xx).

Напомним, что проверка 3-го разряда регистра состояний, содержащего константу 0, с целью определения, равен ли он нулю, всегда дает положительный результат. Это порождает безусловную передачу управления, которая записывается в виде

82xx 0008; GO TO BR (xx).

Микропрограмма. Полный текст интерпретатора приведен в табл. 4.3. В этой таблице есть одна "лишняя" графа (ВЕРТИКАЛЬНАЯ), содержимое которой описано в параграфе 4.5.

Строки микропрограммы с номерами 01–0С описывают процессы выборки команды интерпретируемой машины, пересылки ее в РК для анализа и увеличения содержимого СК на единицу. Строки с номерами 07–0В образуют декодирующее "дерево", которое проверяет биты 15, 14, 13 и 12 регистра команд для определения типа команды и передачи управления в соответствующую часть микропрограммы. При этом используется то обстоятельство, что коды операций адресных команд расположены в диапазоне от 0000 до 1100, т. е. обязательно содержат цифру 0 в 15, 14 и 13 разрядах, а команды ввода-вывода (код операции 1110) отличаются от безадресных команд (код операции 1111) 12 разрядом.

Если выбрана адресная команда, то в строке 0С проверяется заданный в ней вид адресации (прямая или косвенная). При прямой адресации осуществляется переход на строку 1D, а при косвенной –

переход к микрокомандам цикла выборки адреса операнда (строки 0D – 1C) и далее к строке 1D.

В цикле выборки адреса операнда сначала производится выборка в РД этого адреса, а затем – проверка, не хранился ли он в одной из индексных ячеек (ячеек с адресами от 008 до 00F). Если да, то содержимое РД увеличивается на 1, затем пересылается обратно в индексную ячейку и, наконец, восстанавливается (от содержимого РД вычитается 1). Далее начинают выполняться микрокоманды цикла исполнения адресных команд.

В цикле исполнения адресных команд, который начинается с микрокоманды отделения команд переходов, расположенной по адресу 1D, производятся дальнейшее декодирование команды и ее исполнение. После отделения команд переходов осуществляется отделение арифметических команд (SUB, ADD, ADC), команд ISZ и AND и т. д.

Реализация любой адресной команды (впрочем, как и любой другой команды) завершается микрокомандой перехода к циклу прерывания (микрокоманды 8F - 98), где выясняется режим работы ЭВМ и, если машина находится в состоянии "ПРОГРАММА", то проверяется, не требуется ли прервать выполнение программы по запросу внешнего устройства (см. параграф 3.5). Если не требуется, то осуществляется переход к началу микропрограммы (строка 01), т. е. к выборке новой команды интерпретируемой машины. В противном случае организуется переход к подпрограмме обработки прерываний.

Аналогичным образом обрабатываются безадресные команды, команды ввода-вывода и команды, задаваемые нажатием кнопок на пульте управления ЭВМ.

Таблица 4.3

Интерпретатор учебной ЭВМ (микропрограмма

Адрес	Микрокоманды		Комментарии	
	Горизонт.	Верт.	Метка	Действие
Цикл выборки команды				
01	0000 0008	0300	нач	СК ==> БР
02	0004 0000	4001		БР ==> РА
03	0080 0408	0311		ОП(РА) ==> РД, СК + 1 ==> БР
04	0020 0000	4004		БР ==> СК
05	0000 0002	0100		РД ==> БР
06	0010 0000	4003		БР ==> РК
				Определение типа команды
07	880C 8000	AF0C		IF BIT(15,РК) = 0 THEN АДЦ(0C)
08	880C 4000	AE0C		IF BIT(14,РК) = 0 THEN АДЦ(0C)
09	880C 2000	AD0C		IF BIT(13,РК) = 0 THEN АДЦ(0C)
0A	895E 1000	EC5E		IF BIT(12,РК) = 1 THEN БАД(5E)
0B	828E 0008	838E		GOTO B/B(8E)
				Определение вида адресации

Адрес	Микрокоманды		Комментарии	
	Горизонт.	**Верт.**	**Метка**	**Действие**
0C	881D 0800	AB1D	АДЦ	IF BIT(11,РК) = 0 THEN АДР(1D)
Цикл выборки адреса операнда				
0D	0000 0002	0100		РД ==> БР
0E	0004 0000	4001		БР ==> РА
0F	0080 0000	0001		ОП(РА) ==> РД
10	881D 0008	A31D		IF BIT(3,РК) = 0 THEN АДР(1D)
11	891D 0010	E41D		IF BIT(4,РК) = 1 THEN АДР(1D)
12	891D 0020	E51D		IF BIT(5,РК) = 1 THEN АДР(1D)
13	891D 0040	E61D		IF BIT(6,РК) = 1 THEN АДР(1D)
14	891D 0080	E71D		IF BIT(7,РК) = 1 THEN АДР(1D)
15	891D 0100	E81D		IF BIT(8,РК) = 1 THEN АДР(1D)
16	891D 0200	E91D		IF BIT(9,РК) = 1 THEN АДР(1D)
17	891D 0400	EA1D		IF BIT(10,РК) = 1 THEN АДР(1D)
18	0000 0402	0110		РД ==> БР
19	0008 0000	4002		БР ==> РД
1A	0100 0000	0002		РД ==> ОП(РА)
1B	0000 0082	0140		РД + СОМ(0) = РД - 1 ==> БР
1C	0008 0000	4002		БР ==> РД
Цикл исполнения адресных команд				
				Декодирование адресных команд
1D	892D 8000	EF2D	АДР	IF BIT(15,РК) = 1 THEN ПРХ(2D)
1E	0000 0002	0100		РД ==> БР
1F	0004 0000	4001		БР ==> РА
20	8927 4000	EE27		IF BIT(14,РК) = 1 THEN АРФ(27)
21	8824 2000	AD24		IF BIT(13,РК) = 0 THEN A1(24)
22	8857 1000	AC57		IF BIT(12,РК) = 0 THEN JSR(57)
23	8238 0008	8338		GOTO MOV(38)
24	0080 0000	0001	A1	ОП(РА) ==> РД
25	8850 1000	AC50		IF BIT(12,РК) = 0 THEN ISZ(50)
26	8235 0008	8335		GOTO AND(35)
27	0080 0000	0001	АРФ	ОП(РА) ==> РД
28	882B 2000	AD2B		IF BIT(13,РК) = 0 THEN СУМ(2B)
29	8843 1000	AC43		IF BIT(12,РК) = 0 THEN SUB(43)
2A	82B0 0008	83B0		GOTO Р - А(B0)
2B	883C 1000	AC3C	СУМ	IF BIT(12,РК) = 0 THEN ADD(3C)
2C	823F 0000	833F		GOTO ADC(3F)
2D	8830 4000	AE30	ПРХ	IF BIT(14,РК) = 0 THEN УПХ(30)
2E	8847 1000	AC47		IF BIT(12,РК) = 0 THEN BR(47)
2F	82D0 0008	83D0		GOTO Р - П(D0)
30	8833 2000	AD33	УПХ	IF BIT(13,РК) = 0 THEN П1(33)
31	884C 1000	AC4C		IF BIT(12,РК) = 0 THEN BMI(4C)
32	824E 0008	834E		GOTO BEQ(4E)
33	8846 1000	AC46	П1	IF BIT(12,РК) = 0 THEN BCS(46)
34	824A 0008	834A		GOTO BPL(4A)
				Исполнение адресных команд
35	0000 0212	1120	AND	А & РД ==> БР
36	0040 C000	4035		БР ==> A, N, Z
37	828F 0008	838F		GOTO ПРЕ(8F)
38	0000 0010	1000	MOV	А ==> БР
39	0008 0000	4002		БР ==> РД
3A	0100 0000	0002		РД ==> ОП(РА)
3B	828F 0008	838F		GOTO ПРЕ(8F)
3C	0000 0012	1100	ADD	А + РД ==> БР
3D	0040 E000	4075		БР ==> A, C, N, Z
3E	828F 0008	838F		GOTO ПРЕ(8F)
3F	823C 0001	803C	ADC	IF BIT(0,РС) = 0 THEN ADD(3C)
40	0000 0412	1110		А + РД + 1 ==> БР
41	0040 E000	4075		БР ==> A, C, N, Z

Адрес	Микрокоманды		Комментарии	
	Горизонт.	Верт.	Метка	Действие
42	828F 0008	838F		GOTO ПРЕ(8F)
43	0000 0512	1190	SUB	A + COM(РД) + 1 = A - РД ==> БР
44	0040 E000	4075		БР ==> A, C, N, Z
45	828F 0008	838F		GOTO ПРЕ(8F)
46	828F 0001	808F	BCS	IF BIT(0,PC) = 0 THEN ПРЕ(8D)
47	0000 0002	0100	BR	РД ==> БР
48	0020 0000	4004		БР ==> СК
49	828F 0008	838F		GOTO ПРЕ(8F)
4A	838F 0004	C28F	BPL	IF BIT(2,PC) = 1 THEN ПРЕ(8F)
4B	8247 0008	8347		GOTO BR(47)
4C	828F 0004	828F	BMI	IF BIT(2,PC) = 0 THEN ПРЕ(8F)
4D	8247 0008	8347		GOTO BR(47)
4E	828F 0002	818F	BEQ	IF BIT(1,PC) = 0 THEN ПРЕ(8F)
4F	8247 0008	8347		GOTO BR(47)
50	0000 0402	0110	ISZ	РД + 1 ==> БР
51	0008 0000	4002		БР ==> РД
52	0100 0000	0002		РД ==> ОП(РА)
53	858A 8000	DF8F		IF BIT(15,РД) = 1 THEN ПРЕ(8F)
54	0000 0408	0310		СК + 1 ==> БР
55	0020 0000	4004		БР ==> СК
56	828F 0008	838F		GOTO ПРЕ(8F)
57	0000 0402	0110	JSR	РД + 1 ==> БР
58	0010 0000	4003		БР ==> РК
59	0000 0008	0300		СК ==> БР
5A	0008 0000	4002		БР ==> РД
5B	0100 0004	0202		РД ==> ОП(РА), РК ==> БР
5C	0020 0000	4004		БР ==> СК
5B	828F 0008	838F		GOTO ПРЕ(8F)
Продолжение цикла выборки команды декодирование и исполнение безадресных команд				
5E	8861 0800	AB61	БАД	IF BIT(11,РК) = 0 THEN Б0(61)
5F	886C 0400	AA6C		IF BIT(10,РК) = 0 THEN Б1(6C)
60	82E0 0008	83E0		GOTO Р - Б(E0)
61	8867 0400	AA67	Б0	IF BIT(10,РК) = 0 THEN Б2(67)
62	8865 0200	A965		IF BIT(9,РК) = 0 THEN Б3(65)
63	8882 0100	A882		IF BIT(8,РК) = 0 THEN ROL(82)
64	8285 0008	8385		GOTO ROR(85)
65	887B 0100	A87B	Б3	IF BIT(8,РК) = 0 THEN CMA(7B)
66	827E 0008	837E		GOTO CMC(7E)
67	886A 0200	A96A	Б2	IF BIT(9,РК) = 0 THEN Б4(6A)
68	8876 0100	A876		IF BIT(8,РК) = 0 THEN CLA(76)
69	8279 0008	8379		GOTO CLC(79)
6A	8888 0100	A888	Б4	IF BIT(8,РК) = 0 THEN HLT(88)
6B	8287 0008	8387		GOTO NOP(87)
6C	886F 0200	A96F	Б1	IF BIT(9,РК) = 0 THEN Б5(6F)
6D	888A 0100	A88A		IF BIT(8,РК) = 0 THEN EI(8A)
6E	828C 0008	838C		GOTO DI(8C)
6F	8873 0100	A873	Б5	IF BIT(8,РК) = 0 THEN INC(73)
70	0000 0110	1080	DEC	A + COM(0) = A - 1 ==> БР
71	0040 E000	4075		БР ==> A, C, N, Z
72	828F 0008	838F		GOTO ПРЕ(8F)
73	0000 0410	1010	INC	A + 1 ==> БР
74	0040 E000	4075		БР ==> A, C, N, Z
75	828F 0008	838F		GOTO ПРЕ(8F)
76	0000 0200	0020	CLA	0 ==> БР
77	0040 C000	4035		БР ==> A, N, Z
78	828F 0008	838F		GOTO ПРЕ(8F)
79	0001 0000	4080	CLC	0 ==> C

Адрес	Микрокоманды		Комментарии	
	Горизонт.	**Верт.**	**Метка**	**Действие**
7A	828F 0008	838F		GOTO ПРЕ(8F)
7B	0000 0090	1040	СМА	COM(A) ==> БР, инверсия А
7C	0040 C000	4035		БР ==> A, N, Z
7D	828F 0008	838F		GOTO ПРЕ(8F)
7E	8280 0001	8080	СМС	IF BIT(0,PC) = 0 THEN Б6(80)
7F	8279 0008	8379		GOTO CLC(79)
80	0002 0000	40C0	Б6	1 ==> C
81	828F 0008	838F		GOTO ПРЕ(8F)
82	0000 1000	0008	ROL	RAL(A) ==> БР, сдвиг влево
83	0040 E000	4075		БР ==> A, C, N, Z
84	828F 0008	838F		GOTO ПРЕ(8F)
85	0000 0800	0004	ROR	RAR(A) ==> БР, сдвиг вправо
86	0040 E000	4075		БР ==> A, C, N, Z
87	828F 0008	838F	NOP	GOTO ПРЕ(8F)
88	0000 0001	4008	HLT	Останов машины
89	8201 0008	8301		GOTO НАЧ(01)
8A	1000 0000	4800	EI	Разрешение прерывания
8B	8201 0008	8301		GOTO НАЧ(01)
8C	0800 0000	4400	DI	Запрещение прерывания
8D	8201 0008	8301		GOTO НАЧ(01)
Продолжение цикла выборки команды декодирование и исполнение команд ввода-вывода				
8E	0200 0000	4100	В/В	Организация связей с ВУ
Цикл прерывания				
8F	8288 0080	8788	ПРЕ	IF BIT(7,PC) = 0 THEN HTL(88)
90	8201 0020	8501		IF BIT(5,PC) = 0 THEN НАЧ(01)
91	0000 0200	0020		0 ==> БР
92	0004 0000	4001		БР ==> РА
93	0000 0008	0300		СК ==> БР
94	0008 0000	4002		БР ==> РД
95	0100 0400	0012		РД ==> ОП(РА), 1 ==> БР
96	0020 0000	4004		БР ==> СК
97	0800 0000	4400		Запрещение прерывания
98	8201 0008	8301		GOTO НАЧ(01)
Пультовые операции				
				Ввод адреса
99	0000 0040	3000	В/А	КР ==> БР
9A	0020 0000	4004		БР ==> СК
9B	828F 0008	8388		GOTO ПРЕ(88)
				Чтение
9C	0000 0008	0300	ЧТ	СК ==> БР
9D	0004 0000	4001		БР ==> РА
9E	0080 0408	0311		ОП(РА) ==> РД, СК + 1 ==> БР
9F	0020 0000	4004		БР ==> СК
A0	828F 0008	8388		GOTO ПРЕ(88)
				Запись
A1	0000 0008	0300	ЗАП	СК ==> БР
A2	0004 0000	4001		БР ==> РА
A3	0000 0040	3000		КР ==> БР
A4	0008 0000	4002		БР ==> РД
A5	0100 0408	0312		РД ==> ОП(РА), СК + 1 ==> БР
A6	0020 0000	4004		БР ==> СК
A7	828F 0008	8388		GOTO ПРЕ(88)
				Пуск
A8	0000 0200	0020	ПУСК	0 ==> БР
A9	005C E000	4077		БР ==> A, C, N, Z, РА, РД, РК
AA	0400 0000	4200		Сброс флагов ВУ
AB	0800 0000	4400		Запрещение прерывания

Адрес	Микрокоманды		Комментарии	
	Горизонт.	Верт.	Метка	Действие
AC	828F 0008	838F		GOTO ПРЕ(8F)
. . .				
B0			Р - А	Арифметическая команда 7###
. . .				
D0			Р - П	Команда перехода D###
. . .				
E0			Р - Б	Безадресная команда FC##
. . .				
FF				

В системе команд базовой ЭВМ ряд кодов операций зарезервирован для включения новых команд. Это арифметическая команда 7xxx, команда перехода Dxxx и безадресные команды FC00, FD00, FE00 или FF00. Когда при декодировании команды выясняется, что выбрана команда 7xxx, производится передача управления к строке B0. Следовательно, часть микропрограммы, описывающая последовательность микроопераций по реализации этой команды, должна начинаться со строки B0. Сюда можно записать, например, действия по умножению или делению операндов. Часть микропрограммы, реализующая дополнительные безадресные команды, начинается от строки E0. Первые микрокоманды этой части должны осуществить декодирование выбранной команды (определить расширение кода операции: C, D, E или F) и передать управление соответствующим микрокомандам нового куска микропрограммы. Так как память микрокоманд имеет 256 ячеек, то при составлении микропрограмм новых команд можно использовать лишь строки с номерами от AD до FF.

Заключение

Рассмотрим, наконец, что же, собственно говоря, мы называем микропрограммированием. Основная идея изложенного выше подхода сводится к тому, чтобы начать с очень простой аппаратно реализованной машины (микропрограммный уровень базовой ЭВМ имеет лишь две простые команды) и использовать ее для написания интерпретатора более сложной, так называемой интерпретируемой машины. Эта машина, в свою очередь, может выполнять роль интерпретирующей для другой виртуальной (кажущейся) машины и т. д.

4.5. Другие варианты построения микрокоманд

В параграфе 1.5 объяснялось, почему короткие команды лучше, чем длинные: такие команды позволяют сэкономить память и увеличить количество команд, которое может быть извлечено из памяти за одну

секунду. Это справедливо и в отношении микропрограммного уровня. Микрокоманда с одним битом, приходящимся на одну вентильную схему (горизонтальная микрокоманда), привлекательна простотой построения и реализации (единицы и нули, содержащиеся в микрокоманде, непосредственно поступают на вентильные схемы). Однако в случае вычислительной машины сравнительно больших размеров число вентильных схем может составлять несколько сотен и горизонтальная микрокоманда окажется неприемлемо длинной.

Кодируемые поля. Не все возможные комбинации 32 бит можно использовать в рассмотренных ранее микрокомандах базовой ЭВМ (см. табл. 4.2). Так, вентильные схемы 1, 2 и 3; 1 и 19; 23 и 24 и много других групп не могут быть открыты одновременно (см. рис. 4.7). Хотя принципиально нет ограничений на то, чтобы вентильные схемы 18, 19, 20, 21 и 22 были открыты одновременно, в действительности редко возникает необходимость в открытии в данный момент времени более одной из них.

Недопустимость более одного сигнала на левом и правом входах АЛУ, задания сигналов на входы АЛУ и использование в том же такте полученного результата, одновременной записи в БР суммы операндов и результатов их логического умножения – все это позволяет разбить операционную микрокоманду на две и сократить общее число используемых разрядов микрокоманды.

В управляющей микрокоманде сокращение ее разрядов можно обеспечить за счет кодирования полей выбора проверяемого регистра и проверяемого бита в этом регистре, что позволяет сократить УМК до 16 бит, т. е. вдвое.

Один из возможных вариантов 16-разрядных микрокоманд базовой ЭВМ приведен на рис. 4.10.

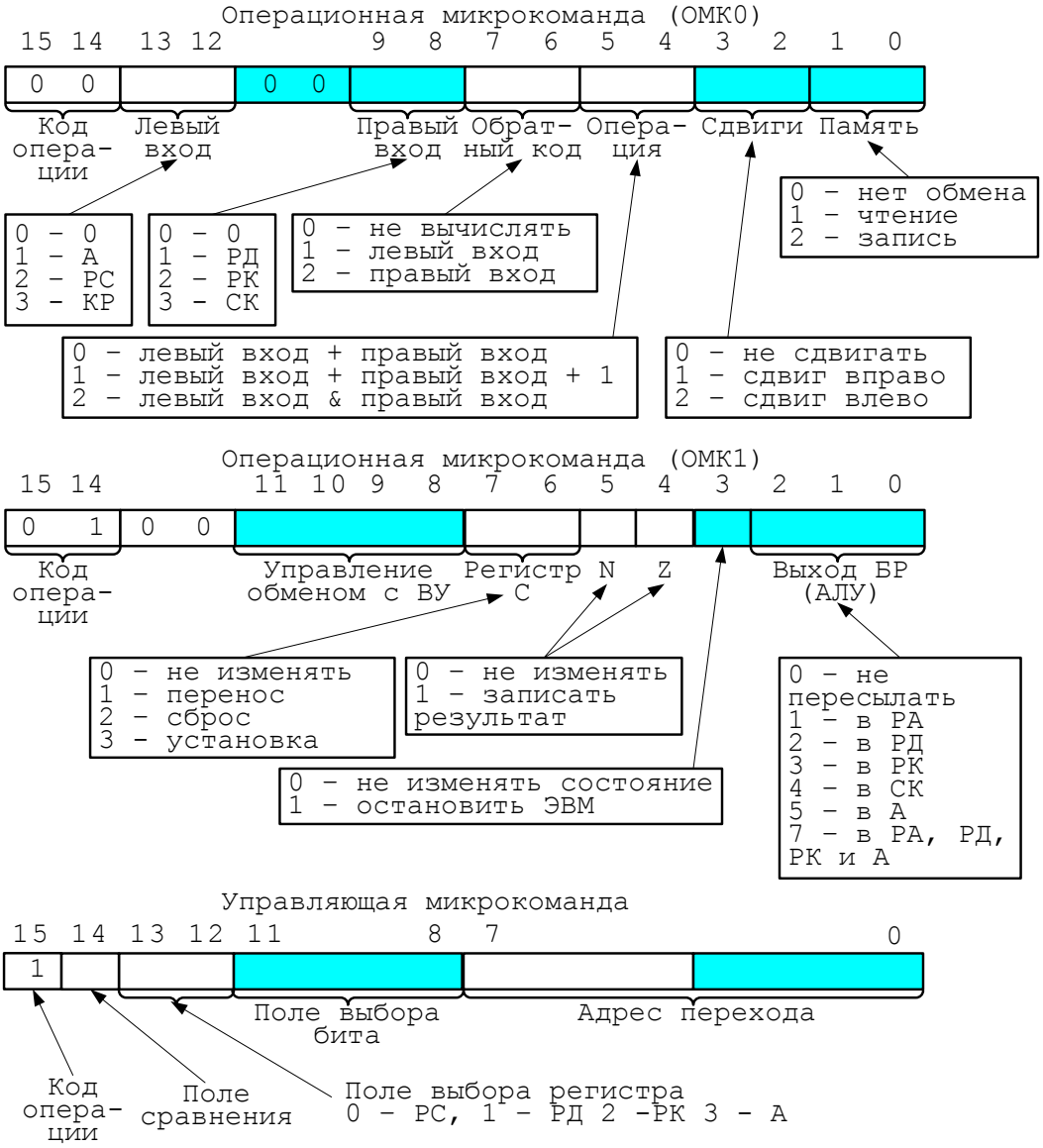


Рис. 4.10. Шестнадцатиразрядные микрокоманды базовой ЭВМ

Здесь почти вся информация закодирована. Например, поле выбора проверяемого бита новой УМК в четыре раза меньше, чем в старой УМК (см. рис. 4.8). Несколько сократились и другие поля (чем длиннее поле, тем больше экономия от кодирования). Однако для декодирования таких сжатых полей требуется использовать специальную схему - дешифратор.

Дешифратор «1 из 2^n» представляет собой комбинационную схему с n входами и 2^n выходами (на рис. 4.11 показаны два дешифратора: «1 из 16», имеющий 4 входа и 2^4=16 выходов, и «1 из 4», имеющий 2 входа и 2^2 = 4 выхода). Каждая выходная линия дешифратора однозначно соответствует одной из 2^n возможных комбинаций входных сигналов.

На рис. 4.11 приведена часть одного из вариантов микропрограммного устройства управления базовой ЭВМ в момент

обработки 16-разрядной управляющей микрокоманды EC5E - IF BIT (12, РК) - 1 THEN БАД (5E).

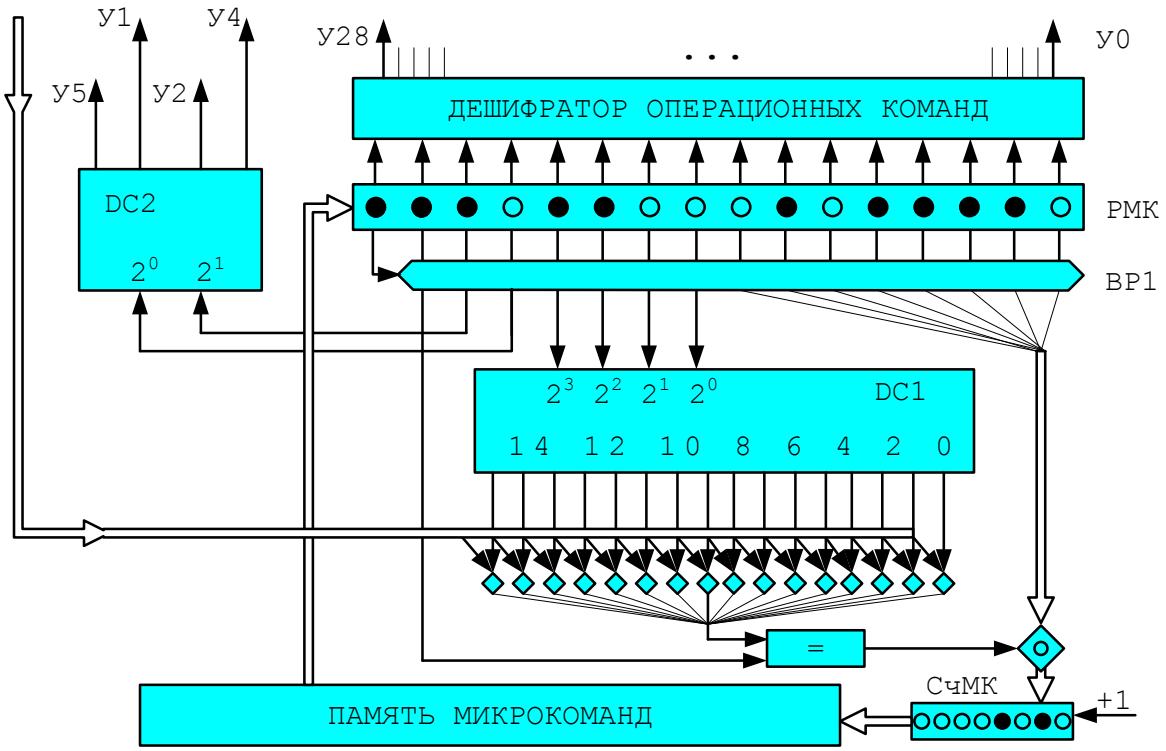


Рис. 4.11. Часть микропрограммного устройства управлении с 16-разрядными Микрокомандами

Значение $(10)_2$, находящееся в поле выбора проверяемого регистра (12- 13 биты УМК), создает, проходя через дешифратор DC2, единичный управляющий сигнал У2. Этот сигнал открывает вентильную схему В2 (см. рис. 4.7), и содержимое РК переписывается в буферный регистр (БР). Значение $(1100)_2$, находящееся в поле выбора проверяемого бита (8-11-й биты УМК), создает на 12-м выходе дешифратора DC1 единичный управляющий сигнал, который используется для выделения и пересылки на сравнивающее устройство 12-го бита БР, т. е. 12-го бита РК. Так как в однобитовом поле сравнения записана 1, то при единичном значении 12-го бита РК в счетчик микрокоманд (СчМК) переписывается адрес перехода 5E (0-7-й биты УМК).

Для декодирования операционных микрокоманд ОМК0 и ОМК1 (см. рис. 4.10) потребуются еще восемь дешифраторов. Однако экономия затрат на память микрокоманд (эта память сокращается вдвое за счет использования 16-разрядных микрокоманд) будет, вероятно, превышать стоимость этих дешифраторов.

Текст интерпретатора базовой ЭВМ, написанный с использованием 16-разрядных микрокоманд, приведен в табл. 4.3 (графа

ВЕРТИКАЛЬНАЯ). Название графы обусловлено тем, что микрокоманды, состоящие из большого числа кодируемых полей, принято называть вертикальными, в отличие от горизонтальных микрокоманд, в которых каждый бит управляет одной вентильной схемой.

Разбиение такта на фазы. Объем микропрограммной памяти можно сократить и при использовании горизонтальных микрокоманд. Для этого надо увеличить их информационную емкость: кодировать в каждой микрокоманде большее число микроопераций. Исполнение же несовместимых во времени микроопераций можно обеспечить разбиением рабочего такта ЭВМ (времени исполнения одной микрокоманды) на несколько подтактов (фаз), в каждом из которых выполняется одно или несколько элементарных действий по реализации микрокоманды.

На рис. 4.12. приведены три первые микрокоманды машинного цикла "Выборка команды" базовой ЭВМ и последовательность их выполнения при двухфазном тактировании (фазы Ф1 и Ф2). По этим новым (более емким) микрокомандам выполняются те же микрооперации, что и по шести первым микрокомандам в табл. 4.3.

Нанопамять. Количество различных по содержанию горизонтальных микрокоманд обычно невелико, но часть из них используется многократно (например, РА = РК, СК = СК+1, СК = РК и другие команды базовой ЭВМ). Это обстоятельство, а также большая длина таких микрокоманд натолкнули разработчиков ЭВМ на мысль о размещении в специальной дополнительной памяти (нанопамяти) только неповторяющихся микрокоманд. Микропрограмма же в этом случае будет состоять лишь из управляющих микрокоманд и адресов ячеек нанопамяти.

Так как количество ячеек нанопамяти невелико, то их адреса имеют малую разрядность, что позволяет уменьшить длину ячеек памяти микрокоманд, где размещаются такие адреса. Это в ряде случаев существенно сокращает общий объем памяти, несмотря на появление нового вида памяти – нанопамяти.

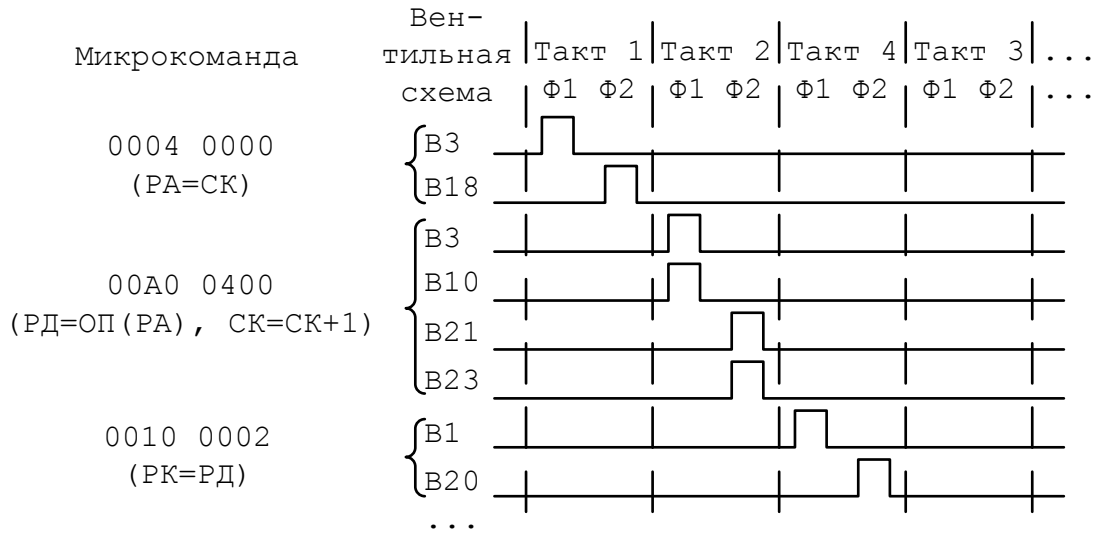


Рис. 4.12. Начало последовательности микроопераций по выборке команды

Сокращение количества и разрядности управляющих микрокоманд. Каждая из управляющих микрокоманд базовой ЭВМ производит проверку лишь одного из битов заданного регистра (см. параграф 4.3). Поэтому для декодирования команды или выявления индексной ячейки иногда приходится использовать длинные последовательности из таких УМК (более 20 УМК).

Уменьшения числа УМК можно добиться несколькими способами:

- модификацией адреса, содержащегося в УМК, путем выполнения определенной логической операции над ним и каким-либо полем заданного регистра.

Это, например, позволяет использовать для декодирования кода операции команды всего лишь одну УМК вместо последовательности из 15-20 УМК;

- проверкой комбинации битов заданного регистра.

Это, например, позволяет использовать для выявления индексной ячейки лишь одну УМК вместо последовательности из 8 УМК.

Уменьшения разрядности УМК в основном добиваются путем замены адреса перехода на более короткое смещение, добавляемое к текущему адресу УМК или вычитаемое из него, и создание нескольких форматов УМК (специализацией УМК). Это позволяет получить УМК, размещаемые в однобайтовых ячейках памяти УМК, что крайне полезно в случае использования нанопамяти.

Микрокоманды различных промышленных ЭВМ чаще всего представляют собой сочетание горизонтальной и вертикальной структур с реализацией тех или иных способов сокращения микропрограммной памяти.

5. МОДЕЛИРОВАНИЕ ПРОЦЕССОВ ФУНКЦИОНИРОВАНИЯ МИКРОЭВМ

5.1. Модельный подход при изучении устройств микропроцессорной техники

Глубокое понимание принципов построения и функционирования сложных систем (каковыми являются устройства микропроцессорной техники) достигается лишь при практической работе с такими системами или их моделями. Даже самое подробное описание работы с базовой ЭВМ (см. параграф 3.1) и детальные иллюстрации процесса выполнения команд (см. например, рис. 2.2) не могут заменить действительной работы на этой ЭВМ, снабженной специальными пультами управления и индикации (см., например, рис. 3.1 и 4.7).

Казалось бы, при сравнительной дешевизне и широком распространении устройств микропроцессорной техники достаточно легко создать стенды для изучения принципов функционирования этих средств вычислительной и управляющей техники. Однако отсутствие доступа к большинству компонентов микропроцессорных устройств (регистров, сумматоров, вентильных схем и т. п.), расположенных внутри одной или нескольких БИС, приводит к необходимости использования для этих целей специально разработанных моделей: физических или функциональных.

Физическая модель – это стенд со структурой изучаемого устройства микропроцессорной техники, построенный на интегральных схемах малой и средней степени интеграции и снабженный средствами для управления режимами работы исследуемого устройства, а также индикации состояний любых его компонентов. В связи с быстрой эволюцией средств микропроцессорной техники массовый выпуск таких специализированных стендов нецелесообразен. Экономически невыгодно и их мелкосерийное производство.

Функциональная модель – это микроЭВМ, в которую введена программа, имитирующая работу исследуемого устройства микропроцессорной техники под воздействием приказов, подаваемых со специально разработанного пульта управления или, например, со штатной клавиатуры микроЭВМ. Состояния компонентов исследуемого устройства отображаются на индикаторах специального пульта или на дисплее микроЭВМ. Таким образом, в качестве технического средства для функционального моделирования цифровых устройств можно

использовать персональную ЭВМ, например бытовой компьютер "Электроника БК-0010" [1] – небольшой ящичек с клавиатурой (370x180x70 мм), подсоединяемый к бытовому телевизору (черно-белому или цветному) и бытовому кассетному магнитофону.

Если существует программа моделирования какого-либо цифрового устройства, составленная на языке персональной ЭВМ "Электроника БК-0010" и записанная на кассетной магнитной ленте, то после ввода такой ленты в машину на экране телевизора будет вычерчена структура исследуемого устройства, например структура контроллера асинхронного последовательного вывода (рис. 5.1). Кроме того, на экране будет воспроизведено (в виде символов 0 и 1) состояние регистров и счетчиков контроллера, а также сигналов на шинах данных и управления.

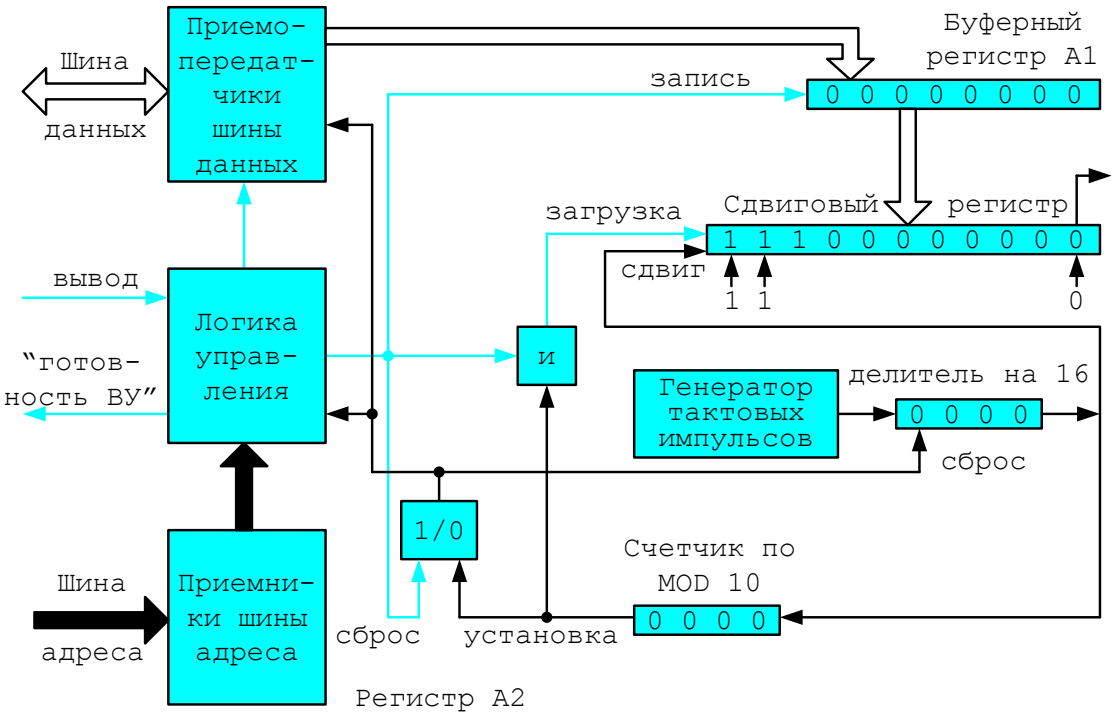


Рис. 5.1. Контроллер асинхронного последовательного вывода

Одновременно с этим определенные клавиши клавиатуры ЭВМ станут органами для изменения информации на шине данных (например, клавиши "0", "1", "7") и линии "Вывод" (например, клавиша "В"). Установив клавишами "0", "7" содержимое выводимого байта данных (содержимое буферного регистра A1), можно (нажатием кнопки "В") дать контроллеру сигнал на преобразование этого байта в последовательность битов и передачу их (вместе со стартовым и стоповым битами) в ВУ. Эти действия легко прослеживаются по изменению (на экране телевизора) содержимого регистров и счетчиков моделируемого контроллера.

Легко заметить, что функциональные модели чрезвычайно универсальны: одни и те же технические средства позволяют производить изучение процесса преобразования информации в любых цифровых устройствах и с любой степенью детализации. Для исследования нового устройства надо только сменить моделирующую программу, программу вывода на экран изображения структуры моделируемого устройства и программу отображения состояния компонентов этого устройства. Основной недостаток функциональных моделей – невозможность исследования работы оригинала при реальных входных сигналах, нагрузках и помехах – присущ и физическим моделям, так как последние строятся на других (по сравнению с оригиналом) элементах, в другом конструктивном исполнении и к ним подключаются лишние устройства (схемы индикации состояний компонентов оригинала).

В соответствии со своим назначением функциональная модель должна правильно и полно отражать состояния лишь индицируемых компонентов оригинала и только в индицируемые моменты времени. Поэтому функциональное моделирование простейших цифровых устройств можно выполнять с помощью программы, которая по сигналам органов управления (например, клавиатуры персональной ЭВМ) выбирает из памяти и "распечатывает" на экране телевизора соответствующие им состояния компонентов оригинала.

Следовательно, при создании функциональной модели надо в первую очередь заботиться о способах ввода управляющих сигналов и вывода информации о состояниях моделируемого устройства, т. е. о факторах, определяющих необходимые уровень и полноту исследования. Та же часть программного обеспечения модели, с помощью которой анализируются текущее ее состояние, наличие сигналов, инициирующих переход в следующее состояние, и определяется новое содержимое индицируемых компонентов оригинала, может быть построена любым удобным (или доступным) для разработчика способом.

5.2. Взаимодействие с функциональной моделью

Если функциональная модель цифрового устройства реализуется с помощью персональной ЭВМ, то взаимодействие с такой моделью проще и дешевле всего организовать через клавиатуру этой ЭВМ и ее дисплей. Аналогичным образом может быть организовано взаимодействие с моделями, построенными на многотерминальной вычислительной системе (в дисплейном классе). Однако такой путь обладает рядом недостатков.

1. Дисплей накладывает серьезные ограничения на объем выводимой информации. Например, при изучении принципа асинхронного последовательного обмена информацией на экране необходимо показать структуру не только передатчика (см. рис. 5.1), но и приемника информации, что трудно сделать, так как на экране можно изобразить лишь 24 строки по 64 или 80 символов в каждой. Кроме того, многие из находящихся в эксплуатации ЭВМ могут выводить на экран лишь буквы, цифры и символы в черно-белом изображении, что резко снижает наглядность создаваемых моделей.

2. Использование штатной клавиатуры персональной ЭВМ (клавиатуры, поставляемой с дисплеем) затрудняет общение с моделью, так как приходится запоминать, на какую функцию запрограммирована та или иная клавиша. Несколько лучше обстоит дело в тех случаях, когда клавиатура имеет ряд функциональных клавиш, программируемых на выполнение нужных действий.

Следовательно, только специально изготовленный пульт управления моделью и индикации ее состояний позволит организовать наиболее эффективное взаимодействие с моделью и получить всю необходимую информацию о ее функционировании.

На рис. 5.2,а приведен внешний вид, а на рис. 5.2,б – структурная схема пульта управления и индикации, используемого для изучения работы базовой ЭВМ. Пульт состоит из набора тумблеров и кнопок (см. рис. 3.1), индикатора матричного газоразрядного (ИМГ-3), контроллера для связи клавиатуры и индикатора с управляющей (моделирующей) ЭВМ.

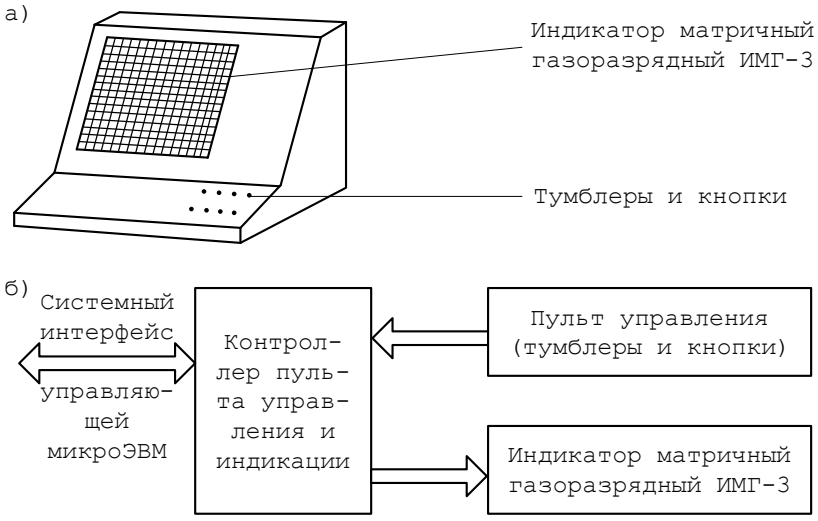


Рис. 5.2. Пульт управления и индикации функциональных моделей микроЭВМ:
а — внешний вид; **б** — структурная схема

ИМГ-3 – газоразрядная панель с устройством управления, которое может зажечь любой из 4096 (64Х64) элементов изображения (ЭИ) размером 3Х3 мм, расположенных на расстоянии 3 мм друг от друга. На панель накладывается маска со структурой базовой ЭВМ (см. рис. 3.1) или ее микропрограммным устройством управления (см. рис. 4.6). В изображениях регистров и других устройств этих масок пробиты круглые отверстия, расположенные над ЭИ панели. Моделирующая программа зажигает ярким зеленым цветом ряд ЭИ, соответствующих единичным состояниям тех или иных разрядов устройств базовой ЭВМ.

Такой пульт можно с успехом использовать для изучения процессов функционирования любых микроЭВМ. Он позволяет индицировать содержимое отдельных устройств для достаточно сложных структур. Органы управления пульта схожи с органами управления мини-ЭВМ, что создает при работе иллюзию общения с аппаратно реализованной ЭВМ, а не с ее программной моделью. Однако изготовление подобного пульта и устройств для его связи с моделирующей ЭВМ – достаточно трудоемкая задача. Поэтому далее будут рассмотрены некоторые приемы, позволяющие использовать для управления работой модели и индикации ее состояний штатные ВУ моделирующей ЭВМ – клавиатуру и дисплей.

Переключатели и кнопки. В составе штатных клавиатур моделирующих персональных ЭВМ отсутствуют двух- и более позиционные переключатели. Поэтому для имитации подобных переключателей моделируемых цифровых систем приходится использовать кнопки (клавиши, не имеющие механической фиксации положения) и часть экрана дисплея моделирующей ЭВМ.

На экране вычерчивается схема переключателя или высвечиваются символы, характеризующие текущее его состояние, а каждое нажатие соответствующей клавиши (кнопки) инициирует изменение состояния переключателя и его изображения на экране. Изменение состояния моделируемого переключателя осуществляется программным путем: получив сигнал от нажатой клавиши, программа анализирует текущее ее "положение" и изменяет его либо на противоположное (двухпозиционный переключатель), либо, например, на следующее по порядку (многопозиционный переключатель).

Первая модель базовой ЭВМ была реализована на персональной ЭВМ 80-х годов прошлого века – "Искра 226" (рис. 5.3)

Рис. 5.3. Профессиональная персональная «Искра 226» [4]

Для задания адреса ячейки памяти или ввода данных в модель базовой ЭВМ, построенной на "Искра 226", используются функциональные клавиши клавиатуры этой персональной ЭВМ (рис. 5.4,б).

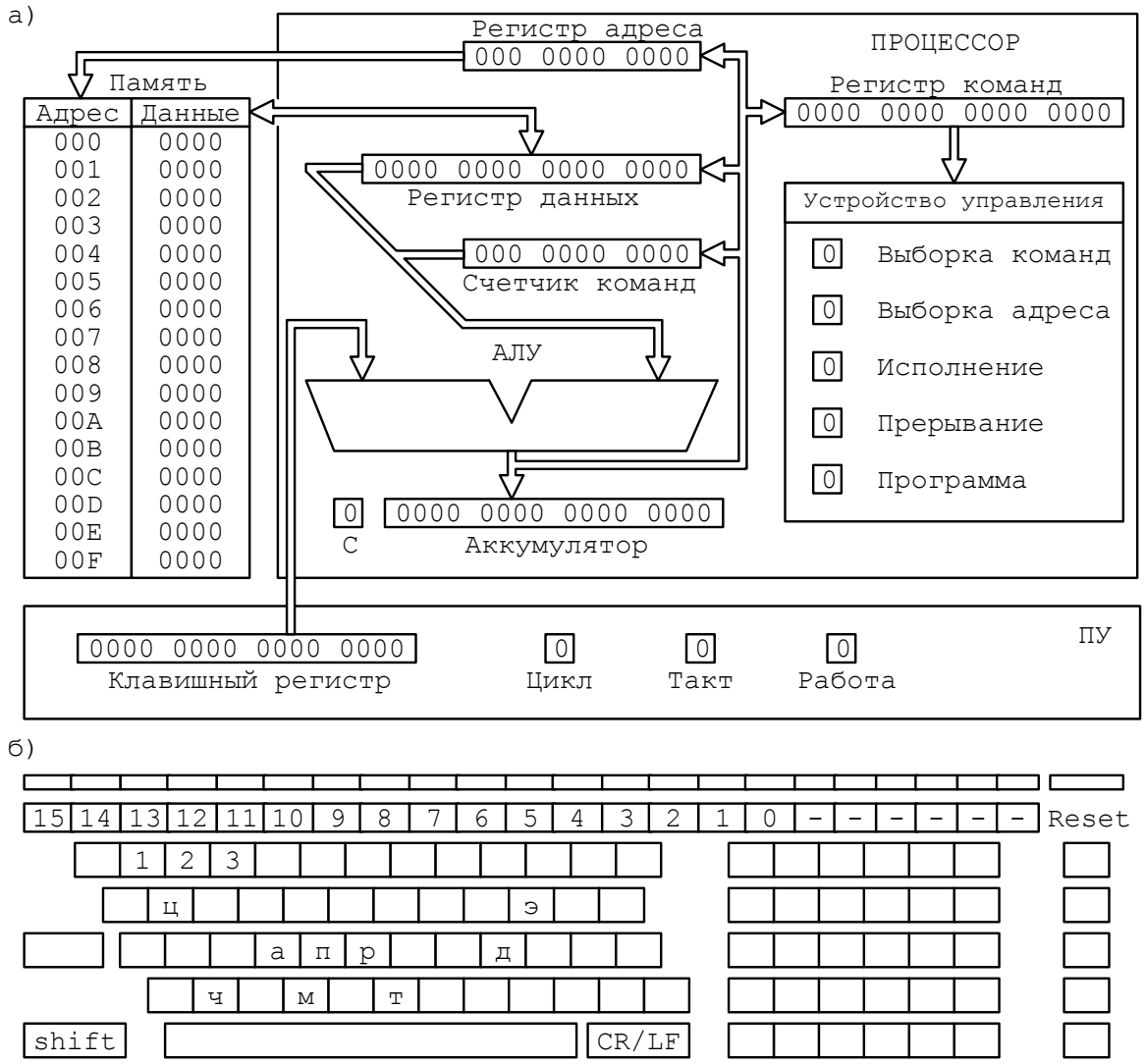


Рис. 5.4. Средства ПЭВМ «Искра 226», используемые для работы с моделью базовой ЭВМ:
а — экран со структурой процессора; *б* — клавиатура

С помощью каждой из клавиш можно изменить состояние одного разряда клавишного регистра, изображенного на экране дисплея (рис. 5.4,а). Если какая-то клавиша находилась в "положении 0" (в соответствующем разряде клавишного регистра высвечивается символ "0"), то после ее однократного нажатия клавиша будет переведена в "положение 1" (символ "0" заменится на символ "1"). Следующее нажатие той же клавиши переведет ее в "положение 0", т. е. вновь инвертирует соответствующий разряд клавишного регистра. Фрагменты моделирующей программы, используемые для осуществления описанных операций, имеют следующий вид:

```
10 DIM R¤1,K¤2,K¤ (16)1,...
15 REM ОПИСАНИЕ РАБОЧЕЙ ПЕРЕМЕННОЙ R¤, А ТАКЖЕ ДВОИЧНОЙ
ПЕРЕМЕННОЙ К¤ И СИМВОЛЬНОГО МАССИВА К¤(), ИСПОЛЬЗУЕМОГО
ДЛЯ СОХРАНЕНИЯ ТЕКУЩЕГО СОДЕРЖИМОГО КЛАВИШНОГО
РЕГИСТРА ("АДРЕС/ДАННЫЕ")
200 KEYIN R¤,240,210:GOTO 300
205 REM ПРОВЕРКА СОСТОЯНИЯ КЛАВИАТУРЫ
210 IF" R¤>HEX(0F)THEN 228:G0SUB’ 50:GOTO 300
215 REM, ОБРАЩЕНИЕ К ПОДПРОГРАММЕ "КЛАВИШНЫЙ РЕГИСТР",
ЕСЛИ НАЖАТА ОДНА ИЗ 16 СПЕЦКЛАВИШ С КОДАМИ 00 – FF
                …
1000 DEFFN’ 50
1005 PEM ПОДПРОГРАММА "КЛАВИШНЫЙ РЕГИСТР"
1810 I=VAL(R¤)+l
1015 PEM ОПРЕДЕЛЕНИЕ НОМЕРА НАЖАТОЙ КЛАВИШИ
1020 IF K¤ (I)="0"THEN1030:K¤ (I)="0":GOTO 1040
1030 K¤ (I)="l"
1835 REM ИНВЕРТИРОВАНИЕ 1-ГО ЭЛЕМЕНТА МАССИВА К¤()
1040 FOR J=1T02:R=0:F0R I=1T08:IF K¤ (I)="0"THEN 1050:R=R+2ˆ(0-1)
1050 NEXT I:BIN(STR(K¤,J,l))=R:NEXT J
1055 REM ОПРЕДЕЛЕНИЕ НОВОГО ЗНАЧЕНИЯ ДВОИЧНОЙ
ПЕРЕМЕННОЙ К¤
1060 PRINT AT (22,0):FOR J=1T04:F0R I=1T04:PRINT K¤ ((J-l)*4+l);
NEXT I:PRINT " ";:NEXT J
1065 REM ПОТЕТРАДНАЯ ПЕЧАТЬ СОДЕРЖИМОГО КЛАВИШНОГО
РЕГИСТРА
1070 RETURN
```

Индикация. Эффективность изучения работы цифровой вычислительно-управляющей) системы в значительной мере зависит от формы представления информации о состоянии ее устройств в

интересующие исследователя моменты времени. Поэтому построение модели следует начинать с создания достаточно подробной структурной схемы исследуемой системы и перечня тех ее устройств и связей между ними, состояние которых должно индицироваться на каждом из отображаемых шагов работы системы. Если изображение всех этих устройств и связей не может быть размещено на экране дисплея, то необходимо либо несколько упростить отображаемую структуру, либо перейти к ее поэтапному исследованию. Последнее даже предпочтительнее, так как чересчур подробная структура сложна для восприятия.

Например, экран персональной ЭВМ "Искра 226" не позволяет разместить изображения базовой ЭВМ, показанного на рис. 3.1. Поэтому исследование способов обмена данными между процессором и внешними устройствами базовой ЭВМ выполняется по структуре, показанной на рис. 5.5, отличающейся от структуры на рис. 5.4, используемой для исследования работы процессора базовой ЭВМ.

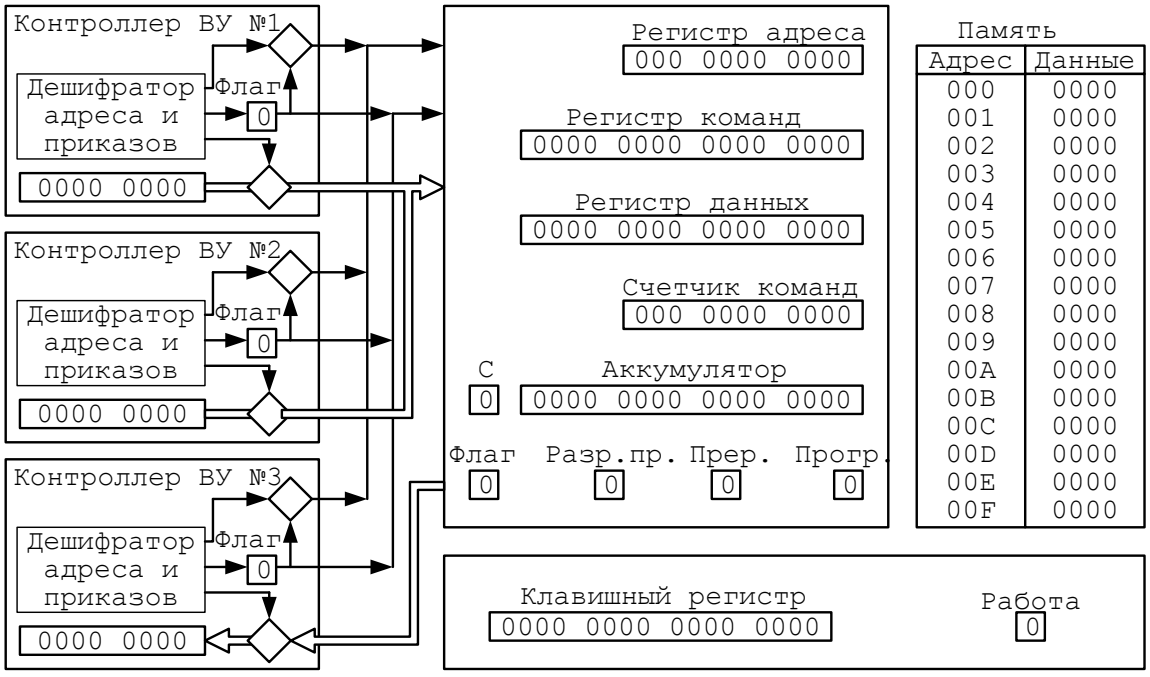


Рис. 5.5. Экран ПЭВМ «Искра 226» со структурой, используемой при изучении обмена данными между процессором и внешними устройствами базовой ЭВМ

При отсутствии символьно-графического дисплея структура исследуемой цифровой системы может быть изображена на маске, накладываемой на экран символьного дисплея. Такую маску лучше всего нарисовать на прозрачной пленке, но можно и на бумаге, в которой затем прорезаются отверстия для индицирования содержимого устройств исследуемой системы.

Это содержимое целесообразнее всего выводить в виде состояния отдельных разрядов индицируемых устройств. Такое состояние можно показать, например, символами "*" и " " (пробел) или "0" и "1". Однако для уменьшения площади изображаемой структуры возможен и вывод в форме шестнадцатеричных чисел (так выводится содержимое ячеек памяти базовой ЭВМ на рис. 5.4 и 5.5).

Потактовую работу системы нагляднее всего иллюстрировать с помощью индикации тех связей, которые устанавливаются между отдельными устройствами системы в каждом из ее состояний. Подобным образом организуется изучение микропрограммного управления базовой ЭВМ (рис. 5.5).

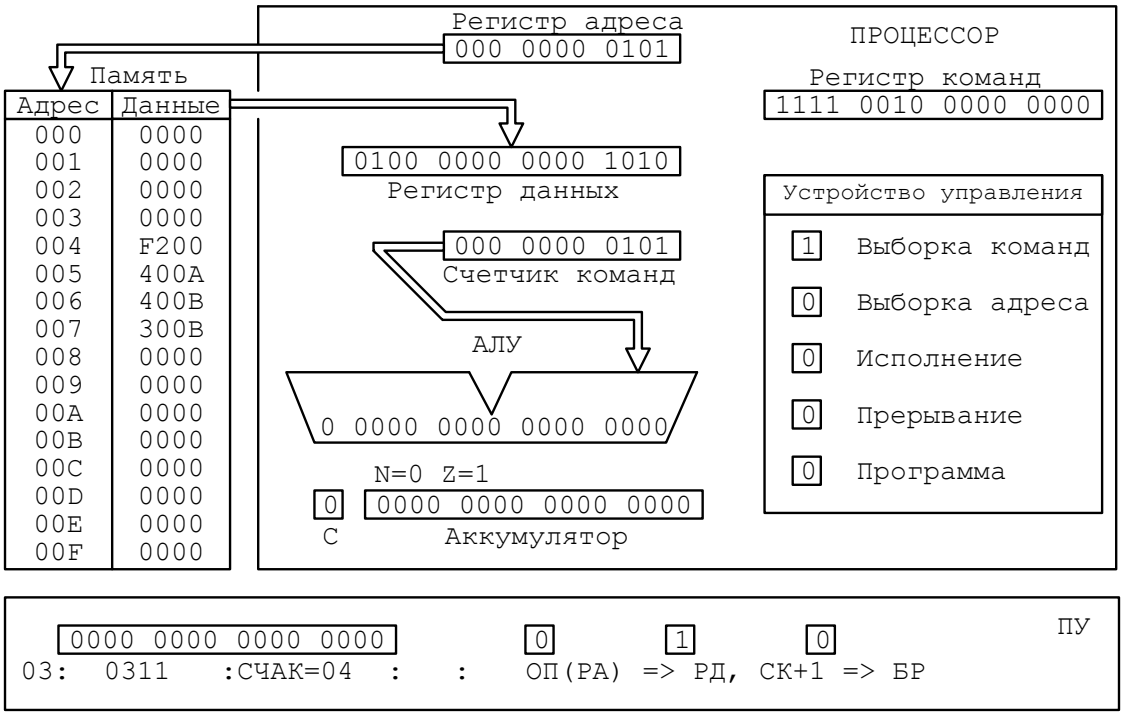


Рис. 5.5. Экран ПЭВМ «Искра 226» со структурой, используемой при изучении микропрограммного управления базовой ЭВМ

5.3. Построение функциональной модели

Назначение функциональной модели (см. параграф 5.1) предопределяет основные режимы ее работы: пошаговый режим и режим достаточно медленного выполнения небольших программ, в процессе которого можно проследить характер изменения состояний всех индицируемых устройств моделируемой вычислительной системы. Следовательно, критерий "минимальное время работы" не является основным при разработке моделирующих программ. Такие программы можно писать на языке высокого уровня даже тогда, когда они будут при работе интерпретироваться на моделирующей ЭВМ.

Создание функциональной модели целесообразно начинать с программ, обеспечивающих взаимодействие с исследователем: программ обработки управляющих сигналов и ввода данных с клавиатуры моделирующей ЭВМ, а также программ вычерчивания изучаемой структуры и вывода состояний ее устройств.

Наименее жесткие требования предъявляются к программам вычерчивания исследуемой структуры, так как эта операция осуществляется перед началом исследования и может занимать достаточно большой промежуток времени (десятки секунд). При создании таких программ для персональных ЭВМ, как правило, не возникает и серьезных ограничений на объем используемой ими памяти (ее незачем экономить при однопрограммном режиме работы ЭВМ).

Программы обработки управляющих сигналов и ввода данных (команд) с клавиатуры моделирующей ЭВМ надо проектировать и размещать среди остальных программ моделирующего комплекса так, чтобы промежуток времени между нажатием какой-либо клавиши и соответствующим изменением рисунка структуры изучаемой системы не превышал долей секунды. В тех же случаях, когда модели требуется больший промежуток времени для обработки приказа, поданного с клавиатуры, целесообразно как-то информировать исследователя о том, что приказ получен и модель приступила к его реализации. Это можно делать путем высвечивания трафарета.

Например, в модели базовой ЭВМ после нажатия клавиш, имитирующих работу кнопок "ВВОД АДРЕСА", "ЗАПИСЬ", "ЧТЕНИЕ", "ПУСК" и "ПРОДОЛЖЕНИЕ", рядом с изображением клавишного регистра выводятся на белом фоне соответствующие надписи: "АДР.", "ЗАП.", "ЧТЕН.", "ПУСК" и "ЖДИТЕ". Трафарет перестает высвечиваться после завершения всех действий по обработке приказа и вывода на экран нового состояния модели.

Вывод состояний устройства исследуемой системы можно осуществлять:

- по мере изменения этих состояний;
- после завершения всех действий по реализации заданного шага (например, микрокоманды или команды);
- в смешанном режиме.

Первый способ целесообразен лишь для простых моделей, в которых на каждом из заданных шагов производится однократное изменение

состояния каких-либо устройств. В противном случае время выполнения шага (или последовательности шагов) существенно возрастет, так как печать информации (даже на экране дисплея) - наиболее длительная из моделирующих операций.

Однако не всегда выгодно запоминать, какие устройства изменяли свое состояние на том или ином шаге, чтобы в конце шага вывести на экран их новое содержимое. Иногда целесообразнее в конце шага выводить на экран содержимое всех устройств модели независимо от того, изменяли ли они на этом шаге свое состояние или не изменяли. Следовательно, выбор того или иного режима вывода информации определяется характером построения модели. В модели базовой ЭВМ используется смешенный режим: содержимое индицируемых регистров процессора и ячеек памяти выводится в конце заданного шага (микрокоманды, машинного цикла или команды), а остальных устройств - по мере изменения их состояний.

Содержание программ, обеспечивающих переработку информации в функциональной модели, определяется не только сложностью оригинала, но и тем, какие устройства оригинала и с каким шагом дискретизации должны индицироваться при работе модели. Так, программы для моделирования микроЭВМ с возможностью исследования ее работы в покомандном режиме значительно проще, чем аналогичные программы, в которых дополнительно предусматривается исследование реализации отдельных команд микроЭВМ по машинным циклам и тактам. Это связано с тем, что функциональная модель должна правильно воспроизводить состояния оригинала лишь в момент вывода на экран содержимого индицируемых устройств. Поэтому в первом случае совсем не обязательно воспроизводить микропрограммный уровень работы оригинала, что позволяет упростить моделирующие программы.

5.4. Исследования, проводимые на функциональной модели

Как было сказано выше, на функциональных моделях нельзя исследовать лишь те аспекты работы оригинала, которые связаны с преобразованием физических сигналов в реальном масштабе времени. Но наиболее удобным их применением является исследование принципов построения и функционирования вычислительных устройств.

Так, модель базовой ЭВМ используется для проведения довольно подробных исследований взаимодействия ее устройств при выборке и исполнении команд [3, 8]. Для этого сначала предлагаются небольшие

последовательности кодов, содержащих команды линейной (табл. 5.1) и разветвляющейся (табл. 5.2) программ, а также коды данных. Последовательность кодов заносится в память базовой ЭВМ (см. параграф 3.1), в счетчик команд заносится пусковой адрес (адрес команды, отмеченной символом " + "), машина устанавливается в режим покомандного исполнения программы, и производится ее пошаговое выполнение. Содержимое всех регистров процессора после исполнения каждой команды заносится в таблицу, аналогичную табл. 3.2, анализируется и комментируется.

Таблица 5.1

Линейная программа			
Ячейки памяти		Ячейки памяти	
Адрес	**Содержимое**	**Адрес**	**Содержимое**
017	0000	01E	301A
018	0255	01F	F200
019	7C99	020	4019
01A	0000	021	101A
01B	+F200	022	301A
01C	4024	023	F000
01D	6018	024	C3CF

Таблица 5.2

Программа с ветвлением			
Ячейки памяти		Ячейки памяти	
Адрес	**Содержимое**	**Адрес**	**Содержимое**
017	54F7	01E	F100
018	+F200	01F	F000
019	4022	020	3017
01A	4023	021	C01F
01B	9020	022	FFD1
01C	F200	023	002F
01D	3017	024	7C99

Затем исследователь должен отразить: назначение программы; реализуемые ею функции (формулы); область представления исходных данных и результатов; расположение в памяти программы, исходных данных и результатов; адреса первой и последней команд программы и т. п. Сходным образом (по готовой или составленной программе) изучается эффективность различных способов организации циклических программ (тех средств базовой ЭВМ, которые позволяют строить такие алгоритмы), работы с подпрограммами (способы передачи параметров) и т. п. Если при этом используется модель, построенная на персональной ЭВМ "Искра 226", то на экране последней вычерчивается структура, показанная на рис. 5.4,а,

С помощью структуры, представленной на рис. 5.4, исследуются алгоритмы обмена информацией с периферийным оборудованием (асинхронного обмена и обмена по прерыванию программы). Исследователь играет роль устройств ввода информации, т. е. устанавливает с помощью клавиатуры коды вводимых символов и дает сигнал о готовности (устанавливает флаг контроллера).

Наконец, с помощью структуры на рис. 5.5 производится изучение функ-ционирования базовой ЭВМ на микропрограммном уровне. Для этого предлагается сначала ознакомиться с потактовым выполнением нескольких команд машины, а затем написать, ввести в память микрокоманд и исполнить несколько новых команд (команд с кодами операций 7, D, FC, FD, FE и FF). Такими командами могут быть команды: загрузки аккумулятора (код операции 7), перехода по четности или нечетности (код операции D). получения дополнительного кода содержимого аккумулятора (код операции FC) и т. п.

Функциональные модели очень удобны для иллюстрации последовательного обмена данными между ЭВМ и ВУ. В них легко показать воздействия помех на искажение передаваемых данных, работу схем контроля по четности (нечетности), ошибки из-за рассинхронизации генераторов при организации асинхронного последовательного обмена и т. п.

Хорошо иллюстрируется и работа серийных контроллеров. Так. для исследования работы параллельного интерфейса контроллера «К» (здесь «К» контроллер 70–80-х годов прошлого века – КР580ВВ55) используется модель со структурой, представленной на рис. 5.6.

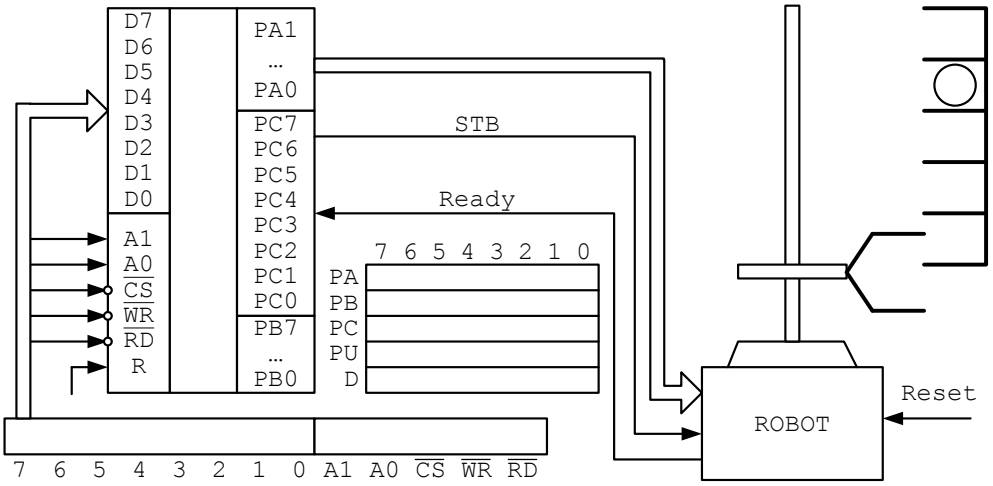


Рис. 5.6. Экран ПЭВМ "Искра 226" со структурой, используемой при изучении микросхемы параллельного интерфейса «К»

С одной стороны, интерфейс «К» связан с процессором, роль которого играет исследователь, а с другой – с моделью складского робота. Через него передаются приказы на выполнение какого-либо действия робота (вверх, вниз, вперед, назад, сжать, разжать). Внутренняя структура интерфейса «К» представлена в виде набора выходных регистров портов ввода-вывода, регистра управления и буферного регистра данных.

Модель параллельного интерфейса программируется исследователем в нужный для обмена режим. Затем исследователь записывает в параллельный интерфейс приказ, который должен быть принят роботом, и два слова, формирующие строб. Приняв строб, робот считывает приказ и выполняет одно из вышеперечисленных действий. Для перемещения детали из одной ячейки в другую исследователю (моделирующему работу процессора) необходимо передать от 20 до 30 8-разрядных слов. Сбои, возникающие из-за неверной команды или других причин, устраняются путем передачи сигнала "Сброс управления робота".

Чрезвычайно просто создавать модели отдельных устройств процессора. На них очень наглядно показывается работа АЛУ, сдвигателей, различных комбинационных схем и т. п.

ПРИЛОЖЕНИЯ

Приложение А

Таблицы перевода чисел из одной системы счисления в другую

Таблица А.1

Сравнение десятичных, двоичных, восьмеричных и шестнадцатеричных кодов чисел

Десятичные	Двоичные	Восьмеричные	Шестнадцатеричные		Десятичные	Двоичные	Восьмеричные	Шестнадцатеричные		Десятичные	Двоичные	Восьмеричные	Шестнадцатеричные
0	0	0	0		11	1011	13	B		22	1 0110	26	16
1	1	1	1		12	1100	14	C		23	1 0111	27	17
2	10	2	2		13	1101	15	D		24	1 1000	30	18
3	11	3	3		14	1110	16	E		25	1 1001	31	19
4	100	4	4		15	1111	17	F		26	1 1010	32	1A
5	101	5	5		16	1 0000	20	10		27	1 1011	33	1B
6	110	6	6		17	1 0001	21	11		28	1 1100	34	1C
7	111	7	7		18	1 0010	22	12		29	1 1101	35	1D
8	1000	10	8		19	1 0011	23	13		30	1 1110	36	1E
9	1001	11	9		20	1 0100	24	14		31	1 1111	37	1F
10	1010	12	A		21	1 0101	25	15		32	10 000	40	20

Таблица А.2

Значения степеней числа 2

n	2^n		n	2^n		n	2^n		n	2^n		n	2^n
0	1		4	16		8	256		12	4 096		16	65 536
1	2		5	32		9	512		13	8 192		17	131 072
2	4		6	64		10	1 024		14	16 384		18	262 144
3	8		7	128		11	2 048		15	32 768		19	524 288

Таблица А.3

Преобразование десятичных чисел в шестнадцатеричные числа

Множитель	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
10	A	14	1E	28	32	3C	46	50	5A
10^2	64	C8	12C	190	1F4	258	2BC	320	384
10^3	3E8	7D0	BB8	FA0	1388	1770	1B58	1F40	2328
10^4	2710	4E20	7530	9C40	C350	EA60	11170	13880	15F90
10^5	186A0	30D40	493E0	61A80	7A120	927C0	AAE60	C3500	DBBA0
10^6	F4240	1E8480	2DC6C0	3D0900	4C4B40	5B8D80	6ACFC0	7A1200	895440
Пример: 1234 = 1000+200+30+4 = $(3E8)_{16}+(C8)_{16}+(1E)_{16}+(4)_{16} = (4D2)_{16}$									

Приложение Б

Кодировки символов

Как уже отмечалось в главе 1, компьютеры хранят всю информацию в виде двоичных байтов, т. е. 8-битовых единиц, способных принимать значение от 0 до 255. Для того, чтобы сохранить в памяти компьютера не числовую, а текстовую информацию, необходимо определить, каким байтом или байтами будет кодироваться каждый символ, который может встретиться в нашем тексте. Такое соответствие между символами и кодирующими их байтами и называется *кодировкой символов* (character set). Нетрудно понять, во-первых, что каждая кодировка разрабатывается для конкретного человеческого языка (точнее, для конкретной письменности), и, во-вторых, что для любого языка таких кодировок можно придумать сколько угодно. Зная человеческую натуру, нетрудно догадаться и о том, что придумают их гораздо больше, чем нужно. Естественно, так и случилось: наиболее развитая на сегодня библиотека функций перекодировки ICU (International Components for Unicode) корпорации IBM поддерживает более 170 различных кодировок.

Кодировки латиницы

Рассмотрим подробнее кодировки тех письменностей, с которыми чаще всего сталкивается российский пользователь, т. е. латиницы и кириллицы. Для латиницы на сегодня используют-ся две основные кодировки: ASCII и EBCDIC. ASCII (American Standard Code for Information Interchange) — это семибитная кодовая таблица (коды символов 00 - 7F или 0 - 127 десятичные), ставшая стандартом для малых и средних компьютеров. В ней байты с шестнадцатеричными кодами 00 — 1F и 7F используются для кодирования управляющих (неотображаемых) символов, а остальные кодируют следующие символы:

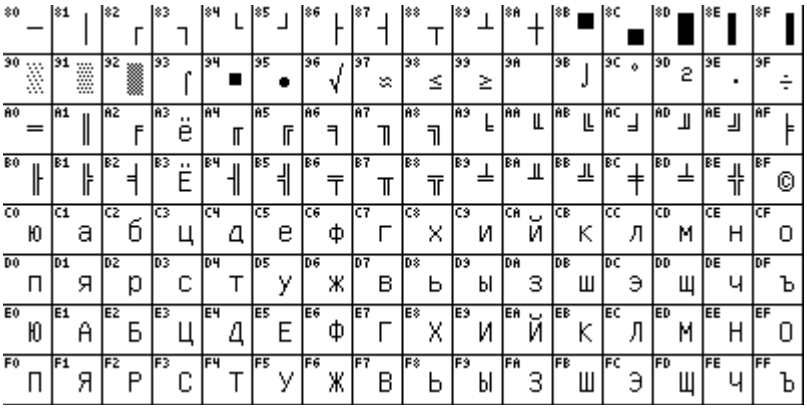
20	21 !	22 "	23 #	24 $	25 %	26 &	27 '	28 (	29)	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	5B [	5C \	5D]	5E ^	5F _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	7B {	7C |	7D }	7E ~	

Кодировки кириллицы

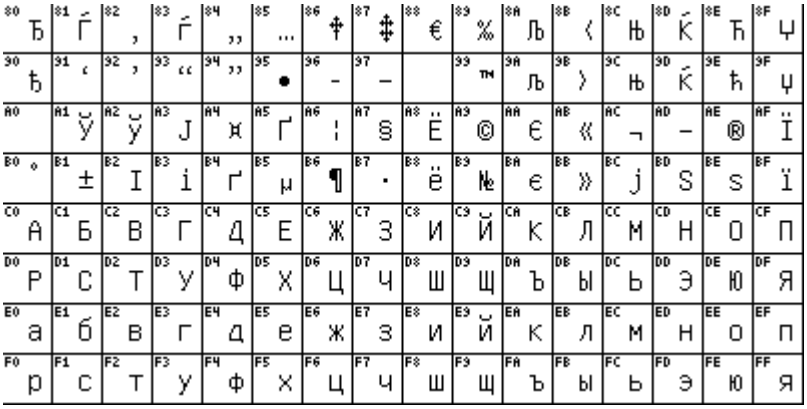
Кодировки «нелатинских» алфавитных письменностей устроены следующим образом. Они кодируются восьмибитовой таблицей (1 байт = 1 символ), т. е. числами 00 - FF (0 - 255 десятичные) так, что младшая половина кодовой таблицы (коды 00 - 7F или 0 - 127 десятичные) совпадает с ASCII, а старшая половина (коды 80 - FF или 128 - 255 десятичные) содержит национальную кодировку, т. е. русские буквы в русских кодовых таблицах, турецкие в турецких и т. д. Такая организация национальных кодовых таблиц позволяет правильно отображать и обрабатывать латинские буквы, цифры и знаки препинания на любом компьютере, независимо от его системных настроек. Именно так, в частности, устроены и русские кодовые таблицы, так что мы можем в дальнейшем рассматривать только старшую их половину.

История русских кодировок — это пример неразберихи, редкостной даже для нашей компьютерной действительности. Советские стандартизирующие организации принимали ГОСТы, производители компьютеров (Apple) и операционных систем (Microsoft) их дружно игнорировали и вводили собственные кодировки. В результате мы получили наследство из четырех разных ГОСТов, две кодировки от Microsoft (для DOS и для Windows) и кодировку от Apple для Mac'ов (все, естественно, несовместимые между собой).

К счастью, сегодня нет нужды подробно описывать все эти кодировки, поскольку в Рунете выжили только две из них. Первая — это КОИ8-Р (КОИ означает Код для Обмена и обработки Информации, ‘Р’ отличает русскую кодовую таблицу от украинской, КОИ8-У. Эта кодировка имеет вид:



КОИ8-Р является стандартом de facto для всех служб Интернета, кроме WWW. В частности, все службы электронной почты и новостей Рунета работают в этой кодировке. Что касается Инетернета, то здесь ситуация сложнее. Дело в том, что более 90% клиентских компьютеров Сети работает под управлением Windows разных версий. Windows использует собственную кодировку русских букв, которую принято назвать по номеру кодовой страницы Windows-1251 или CP1251:



Поскольку текстовые редакторы и средства разработки HTML-страниц в Windows работают в этой кодировке, абсолютное большинство Веб-документов Рунета хранится в кодировке Windows-1251.

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

О кафедре

Кафедра ВТ СПбГУ ИТМО создана в 1937 году и является одной из старейших и авторитетнейших научно-педагогических школ России.

Первоначально кафедра называлась кафедрой математических и счетно-решающих приборов и устройств и занималась разработкой электромеханических вычислительных устройств и приборов управления. Свое нынешнее название кафедра получила в 1963 году.

Кафедра вычислительной техники является одной из крупнейших в университете, на которой работают высококвалифицированные специалисты, в том числе 8 профессоров и 15 доцентов, обучающие около 500 студентов и 30 аспирантов.

Кафедра имеет 4 компьютерных класса, объединяющих более 70 компьютеров в локальную вычислительную сеть кафедры и обеспечивающих доступ студентов ко всем информационным ресурсам кафедры и выход в Интернет. Кроме того, на кафедре имеются учебные и научно-исследовательские лаборатории по вычислительной технике, в которых работают студенты кафедры.

Чему мы учим

Традиционно на кафедре ВТ основной упор в подготовке специалистов делается на фундаментальную базовую подготовку в рамках общепрофессиональных и специальных дисциплин, охватывающих наиболее важные разделы вычислительной техники: функциональная схемотехника и микропроцессорная техника, алгоритмизация и программирование, информационные системы и базы данных, мультимедиа-технологии, вычислительные сети и средства телекоммуникации, защита информации и информационная безопасность. В то же время, кафедра предоставляет студентам старших курсов возможность специализироваться в более узких профессиональных областях в соответствии с их интересами.

Специализации на выбор

Кафедра ВТ ИТМО предлагает в рамках инженерной и магистерской подготовки студентам на выбор по 3 специализации.

1. Специализация в области информационно-управляющих систем направлена на подготовку специалистов, умеющих проектировать и разрабатывать управляющие системы реального времени на основе средств микропроцессорной техники. При этом студентам, обучающимся по этой специализации, предоставляется уникальная возможность участвовать в конкретных разработках реального оборудования, изучая

все этапы проектирования и производства, вплоть до получения конечного продукта. Для этого на кафедре организована специальная учебно-производственная лаборатория, оснащенная самым современным оборудованием. Следует отметить, что в последнее время, в связи с подъемом отечественной промышленности, специалисты в области разработки и проектирования информационно-управляющих систем становятся все более востребованными, причем не только в России, но и за рубежом.

2. Кафедра вычислительной техники - одна из первых, начавшая в свое время подготовку специалистов в области открытых информационно-вычислительных систем. Сегодня студентам, специализирующимся в этой области, предоставляется возможность изучать и осваивать одно из самых мощных средств создания больших информационных систем - систему управления базами данных Oracle. При этом повышенные требования, предъявляемые к вычислительным ресурсам, с помощью которых реализуются базы данных в среде Oracle, удовлетворяются за счет организации на кафедре специализированного компьютерного класса, оснащенного мощными компьютерами фирмы SUN Microsystems, связанными в локальную сеть кафедры. В то же время, студенты, специализирующиеся в данной области, получают хорошую базовую подготовку в области информационных систем, что позволяет им по завершению обучения успешно разрабатывать базы данных и знаний не только в среде Oracle, но и на основе любых других систем управления базами данных.

3. И, конечно же, кафедра не могла остаться в стороне от бурного натиска вычислительных сетей и средств телекоммуникаций в сфере компьютерных технологий. Наличие высокопрофессиональных кадров в данной области и соответствующей технической базы на кафедре (две локальные вычислительные сети, объединяющие около 70 компьютеров и предоставляющие возможность работы в разных операционных средах – Solaris, Linux, Windows), позволило организовать подготовку специалистов по данному направлению, включая изучение вопросов компьютерной безопасности, администрирования, оптимизации и проектирования вычислительных сетей.

СПИСОК ЛИТЕРАТУРЫ

1. Бытовая персональная микроЭВМ «Электроника БК0010»/С.М.Косенков, А.Н.Полосин, З.А.Счепицкий и др.// Микропроцессорные средства и системы. 1985 №1. С. 22-25.
2. Введение в микроЭВМ/С.А.Майоров, В.В.Кириллов, А.А.Приблуда. Л.: Машиностроение. Ленинградское отделение, 1988. 304 с.: ил.
3. Методические указания к лабораторным работам по базовой ЭВМ. Л.: Институт точной механики и оптики, 1986, 54 с.
4. Профессиональные персональные ЭВМ «Искра 226»/С.Н.Абрамович, В.В.Бойко, Б.П. Бутурин и др.// Микропроцессорные средства и системы. 1985 №2. С. 29-36.
5. Соучек Б. Микропроцессоры и микроЭВМ/Пер. с англ. Под ред. А.И.Петренко. М.: Сов. Радио, 1979, 520 с.
6. Таненбаум Э. Многоуровневая организация ЭВМ/ Пер. с англ. Под ред. В.К.Потоцкого и М.Б.Игнатьева. М.: Мир, 1979. 547 с.
7. Труды института инженеров по электронике и радиоэлектронике (ТИИЭР)/ Пер. с англ. М: Мир, 1984. Т.72. № 3 (Тематический выпуск «Персональные ЭВМ»). 187 с.
8. Учебно-методическое пособие к лабораторным работам по дисциплинам «Информатика» и «Введение в специальность». – СПб: СПбГУ ИТМО, 2010. – 120 с.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Ф

Ш

Э

Я

Оглавление

Кириллов Владимир Васильевич

АРХИТЕКТУРА БАЗОВОЙ ЭВМ
Учебное пособие

В авторской редакции
Дизайн Кириллов В.В., Приблуда А.А.
Верстка Кириллов В.В., Приблуда А.А., Лемешев А.С.
Редакционно-издательский отдел Санкт-Петербургского государственного университета информационных технологий, механики и оптики
Зав. РИО Н.Ф. Гусарова
Лицензия ИД № 00408 от 05.11.99
Подписано к печати
Заказ №
Тираж

Отпечатано на ризографе

Редакционно-издательский отдел
Санкт-Петербургского государственного
университета информационных технологий,
механики и оптики

197101, Санкт-Петербург, Кронверкский пр., 49