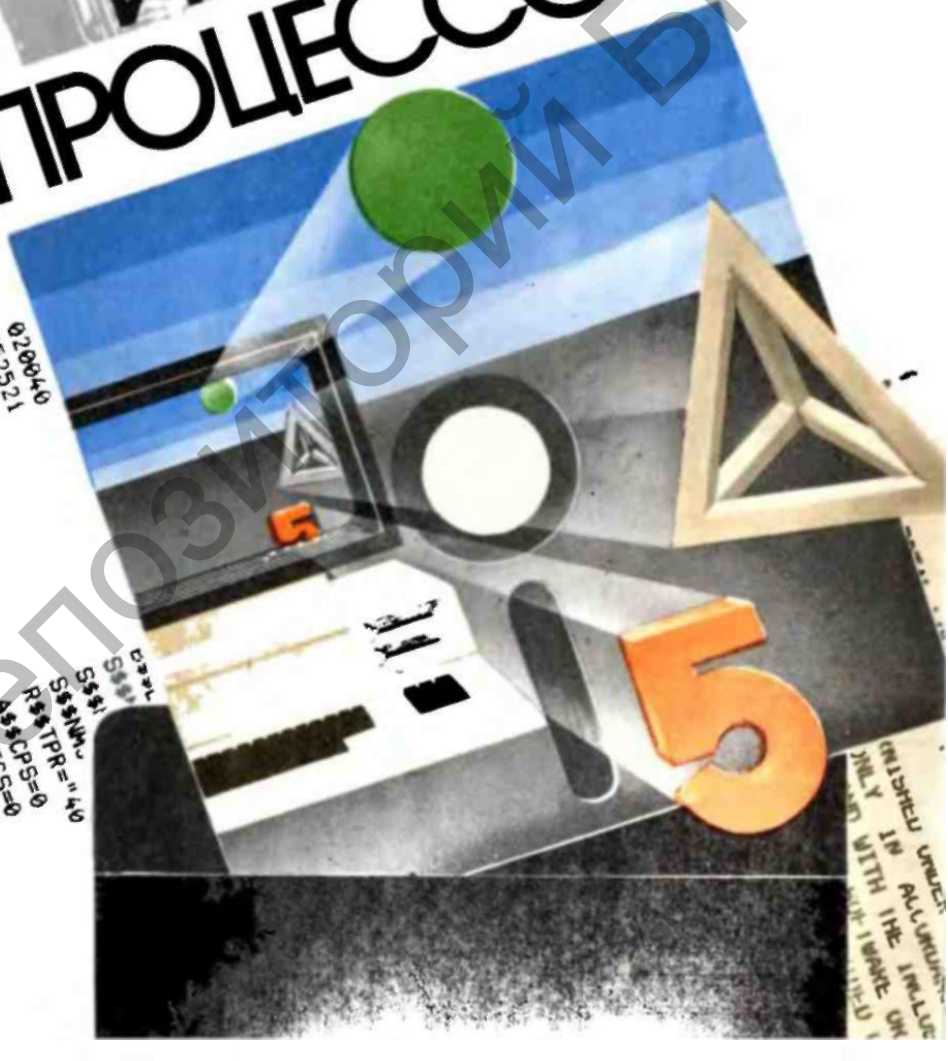
Е. В. Бильдюкевич
В. Л. Гурачевский
С. С. Шушкевич

ЭВМ и микропроцессор



Е. В. Бильдюкевич
В. Л. Гурачевский
С. С. Шушкевич

ЭВМ и микропроцессор

Книга для учащихся

Минск
«Народная асвета»
1990

ББК 32.973
Б61

Рецензенты:
А. Н. Останин, д-р техн. наук,
Л. П. Матюшков, канд. техн. наук

Бильдюкевич Е. В. и др.
Б61 ЭВМ и микропроцессор/Е. В. Бильдюкевич, В. Л. Гурачевский, С. С. Шушкевич: Кн. для учащихся.— Мн.: Нар. асвета, 1990.— 207 с.: ил.
ISBN 5-341-00159-1.

В книге последовательно и доступно излагаются сведения, необходимые для понимания устройства, работы и возможностей применения микропроцессоров и ЭВМ.

Адресуется учащимся старших классов.

4802030000—132
Б————————159—88 **ББК 32.973**
М303(03)—90

ISBN 5-341-00159-1

ПРЕДИСЛОВИЕ, КОТОРОЕ НУЖНО ПРОЧЕСТЬ

Мы стремились сделать книгу доступной для всех тех, кто хочет разобраться в устройстве и применениях микропроцессоров и ЭВМ, но не имеет никаких специальных знаний ни по программированию, ни в области электроники и вычислительной техники.

Но перед вами не развлекательная книга, которую можно читать через строчку. В начале работы над книгой у нас было желание назвать ее «Микропроцессор — это очень просто!», по аналогии с хорошей серией книг по радиоэлектронике, однако позднее мы решили поступить более честно по отношению к читателю, предупредив его, что мир вычислительной техники не только увлекателен и многообразен, но и сложен. Современные ЭВМ и их программное обеспечение по праву относят к самым трудоемким созданиям человеческого разума.

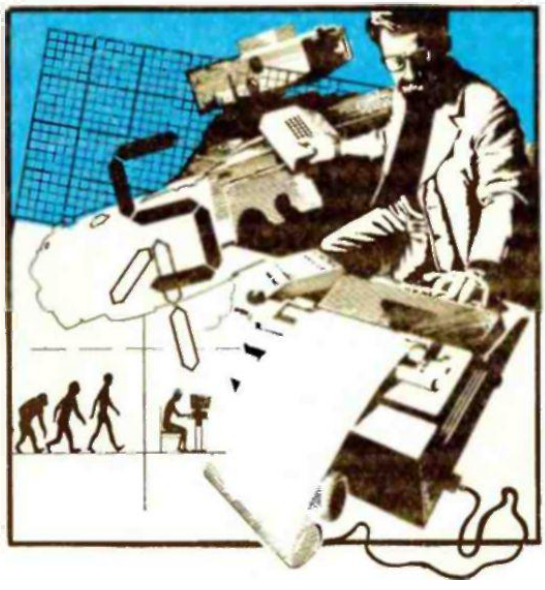
Идеальным будет положение тех читателей, у которых есть возможность поработать с ЭВМ типа ДВК, БК-0010, УК НЦ, СМ-4 или одной из целого ряда других машин, обладающих схожим набором команд. Мы надеемся, что в ходе такой работы поможет прилагаемый к книге справочник, содержащий необходимую программисту информацию об этих ЭВМ.

В книге не нашли отражения вопросы программирования на языках высокого уровня, таких, как Бейсик, Паскаль, Фортран. По этой теме есть достаточное количество литературы.

Желающий понять, как устроена ЭВМ и как с ней работать, должен усвоить ряд понятий и терминов. Многие из них заимствованы из повседневной жизни, но постепенно приобрели качественно иной смысл. Попытка незамедлительного и строгого их определения при первом упоминании таит опасность испугать неподготовленного читателя, что вряд ли позволит дочитать ему книгу до конца, поэтому если вы столкнетесь с понятиями или словами, смысл которых не совсем ясен, не расстраивайтесь и идите дальше. Это не станет непреодолимым препятствием для понимания последующего материала, а незнакомые термины обязательно будут пояснены дальше. Путеводитель по сокращениям и аббревиатурам находится на форзацах.

И еще одна просьба — внимательно читайте подписи к рисункам: они не дублируются в тексте и необходимы для понимания книги.

Авторы



1.1. ЭВМ и вычисления. 1.2. ЭВМ как устройство обработки информации. 1.3. ЭВМ в физическом эксперименте. 1.4. ЭВМ и управление. 1.5. ЭВМ и автоматизация производства. 1.6. ЭВМ в технике. Испытания конструкций и моделирование. 1.7. САПР. 1.8. Автоматизация работы служащих.

ДЛЯ ЧЕГО НУЖНЫ ЭВМ?

Любая ЭВМ состоит из трех основных узлов: процессора, оперативного запоминающего устройства (ОЗУ) и системы ввода-вывода.

Процессор управляет работой всех остальных узлов, производит обработку данных, например выполняет арифметические операции, определяет большее из двух чисел, направляет данные во вполне определенное место ОЗУ и т. п.

ОЗУ — это своего рода склад с возможностью очень быстрого доступа ко всем хранящимся в нем данным, промежуточным и конечным результатам вычислений. В ОЗУ хранится и программа работы, которая воспринимается процессором и определяет характер всех производимых ЭВМ действий.

Система ввода-вывода предназначена для обмена данными между ЭВМ и внешним миром. В зависимости от решаемых задач к ЭВМ могут подключаться различные внешние (периферийные) устройства: печатающие *(принтеры)*, рисующие *(графопостроители*, или «плоттеры»), запоминающие *(внешняя память)*, для диалога с человеком *(терминалы, дисплеи)*, различного рода датчики и исполнительные механизмы. Система ввода-вывода определяет порядок взаимодействия всех внешних устройств с процессором и ОЗУ.

Микропроцессор — это процессор, выполненный в виде одной или нескольких соединенных между собой интегральных схем. ЭВМ, изготовленная на основе микропроцессора, называется микро-ЭВМ. Она выгодно отличается от своих немикропроцес-

сорных родственников габаритами, стоимостью, надежностью и энергопотреблением.

Многие из применений ЭВМ являются следствием того, что машины умеют считать намного быстрее, чем человек.

1.1. ЭВМ и вычисления

Задумывались ли вы когда-нибудь над тем, как получены таблицы элементарных функций типа тригонометрических, логарифмов и т. п.? Принципиальных трудностей на пути вычисления их значений, конечно же, нет. В любом справочнике по математике можно найти, например, формулы для вычисления синуса острого угла

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots,$$

где x — величина угла в радианах.

Чем больше взято членов этого ряда, тем выше точность результата. А теперь попробуйте оценить время, необходимое для составления одной страницы таблицы типа известных всем таблиц Брадиса с помощью электронного калькулятора, способного выполнять сложение, вычитание, умножение и деление.

Даже в том случае, когда калькулятор считает практически мгновенно, на выполнение одной арифметической операции с учетом ввода чисел и считывания результата потребуется 5, а то и 10 секунд. Так что вряд ли следует удивляться, что составление таблиц некоторых функций занимало у людей большую часть жизни!

— Постойте,—скажете вы,—эта работа давным-давно кем-то проделана и хватит ворошить прошлое.

Увы, нет! Современная наука и техника повседневно сталкивается с задачами как можно более точного вычисления все новых и новых, зачастую весьма сложных функций, решение которых в приемлемые сроки не под силу даже большим коллективам людей, если они не вооружены ЭВМ.

А вот другой пример: в 60-х годах экономисты оценили ежегодный объем вычислений, необходимых для эффективного управления народным хозяйством страны. Он составил 10^{17} операций! Здесь уже не обойтись без самых быстродействующих ЭВМ.

Приведя подобного рода пример, уместно поставить вопрос, а как осмыслить итоги таких вычислений. Ведь даже если одна тысячная часть результатов будет напечатана, то это составит почти по полмиллиона чисел на каждого жителя нашей страны. Оказывается, и в этом случае выручает ЭВМ, так как ЭВМ занимается не только вычислениями, но и *обработкой информации* вообще, а вычисления — лишь составная часть этого более широкого понятия.

1.2. ЭВМ как устройство обработки информации

Рассмотрим типичную для ЭВМ задачу расчета заработной платы. На первый взгляд работа сводится к умножению суточной зарплаты каждого служащего на число проработанных дней. Но будет ли ЭВМ заниматься только вычислениями? Конечно, нет. Машина должна будет найти фамилию служащего, разыскать величину его заработной платы, выяснить, не находится ли он в командировке или же отсутствовал по болезни, сверить полученные данные с проработанным этим служащим числом дней и лишь затем произвести операцию умножения. Наконец, полученный результат необходимо отпечатать в нужных позициях ведомости. Даже при решении этой простой задачи основные «усилия» ЭВМ связаны не с вычислениями, а с обработкой информации, под которой можно понимать получение результата из некоторого набора исходных данных.

В такой роли ЭВМ правильнее было бы назвать электронным устройством обработки информации. Однако нетрудно предложить и более точное название — *универсальное устройство обработки информации,* так как ЭВМ способна осуществить произвольную обработку информации, только бы было известно, как ее совершить.

Чтобы заставить машину сыграть новую роль, нужно написать новую *программу* ее работы, т. е. понятную для ЭВМ последовательность приказов (команд), выполнение которых приведет к решению поставленной задачи. Примечательно, что для решения бесконечного множества различных задач обработки информации машина должна уметь выполнять не очень большое количество команд.

Закономерно поставить вопрос, в какой форме должна быть представлена информация, чтобы ЭВМ могла заниматься ее обработкой?

Ответ на этот вопрос изменяется с течением времени. Первые ЭВМ были способны обрабатывать лишь данные, представленные в виде цифр, а сегодня имеются ЭВМ, которые воспринимают и обрабатывают даже человеческую речь. Однако для ЭВМ всех поколений справедливо утверждение о том, что процессор и *запоминающее устройство* (ЗУ) воспринимают информацию только в виде чисел, а все эти числа в свою очередь представлены наборами только из двух символов — *нуля* и *единицы.* Не следует думать, что это ограничивает область применения ЭВМ. Ведь хорошо известно, что по телеграфу можно передать сообщение произвольного содержания и что для этих целей уже в первой половине прошлого века любые послания научились переводить на язык точек и тире, т. е. передавать при помощи только двух символов.

Есть немало поучительных историй об изобретательности физиков-экспериментаторов. В числе их находок — очистка труб оптических приборов от паутины с помощью кошки, получение высокоточного параболического зеркала из вращающегося цилиндра, наполненного ртутью, создание бассейна, в котором могут плавать каменные глыбы, и т. п.

Для решения конкретных задач физического эксперимента были созданы электронные приборы и устройства, которые, как выяснилось впоследствии, оказались весьма совершенными *специализированными* (не универсальными) ЭВМ.

Рассмотрим один такой пример. Предположим, что необходимо измерить энергию гамма-квантов, испускаемых радиоактивным источником.

Существует ряд способов преобразования такой энергии в величину электрического импульса. Для этого гамма-квант должен отдать всю свою энергию устройству для его регистрации — детектору. В действительности же только часть квантов взаимодействует с детектором таким образом, а остальные отдают ему лишь некоторую долю своей полной энергии. Только по этой причине попытка измерить энергию гамма-квантов путем определения величины отдельных электрических импульсов на выходе детектора может оказаться безуспешной.

Ситуация осложняется и побочными явлениями, например тем, что гамма-кванты испускаются веществом в непредсказуемые случайные моменты времени и не исключено одновременное попадание в детектор двух и даже нескольких квантов, что, в свою очередь, приведет к появлению на его выходе «дефектных» импульсов. Они могут быть ошибочно восприняты как результат регистрации высокоэнергетического гамма-кванта.

Разобраться в действительном положении дел помогает анализ очень большого числа событий или так называемый *амплитудный анализ*. Для этого необходимо рассортировать все импульсы по величине независимо от времени их регистрации.

Сначала — в 40-х годах нашего столетия - для этой цели применялись своего рода электромеханические пушки, стрелявшие небольшими стальными шариками тем дальше, чем больше величина поданного на пушку электрического импульса. Если в течение эксперимента шарики собрать в кассету типа показанной на рисунке 1.1, то нетрудно построить график (точнее — гистограмму) распределения импульсов по величине, которую логично назвать *амплитудным распределением*, или спектром амплитуд.

Чтобы из подобного рода гистограммы можно было сделать правильные выводы, нужно получить ее с достаточно большой точностью.

Что это означает?

Если параметры тракта регистрации (детектора, усилителя и самой пушки) стабильны, то точность определения высоты каждой

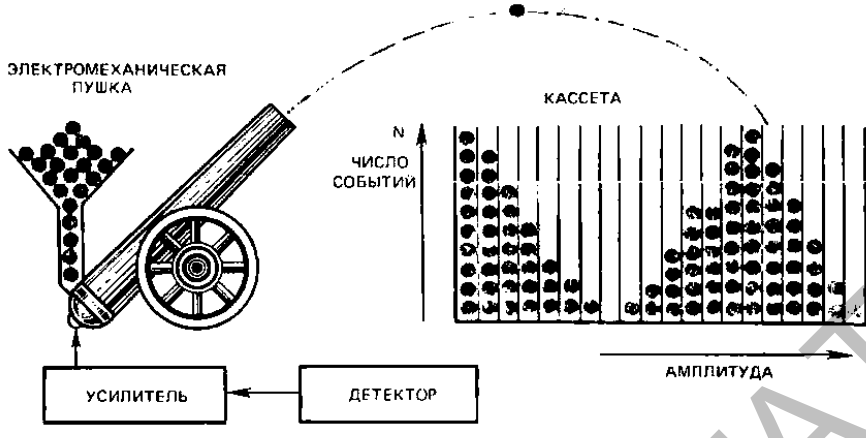


Рис. 1.1. Электромеханическая пушка сортирует импульсы детектора по амплитудам. Каждая ячейка соответствует некоторому интервалу амплитуд, а весь диапазон амплитуд разбит на конечное число таких интервалов (каналов). Дискретны и ординаты «графика», они равны числу шариков в канале. Распределения такого типа принято называть гистограммами.

ступени гистограммы зависит лишь от числа событий, зарегистрированных в канале. На рисунке это соответствует числу шариков в одной ячейке кассеты. При числе шариков N ошибка (ее называют статистической) составляет $\pm\sqrt{N}$. В результате для определения высоты ступеньки с точностью до 1 % число отсчетов в канале должно быть не менее 10 000!

В такой ситуации с электромеханической пушкой далеко не уедешь. Нужна ЭВМ, так как она может сортировать не десятки, как пушка, а сотни тысяч и даже миллионы импульсов в секунду.

На ЭВМ можно возложить и многие другие функции: периодическую проверку исправности тракта регистрации, поиск наиболее эффективных условий измерения, изменение хода эксперимента по полученным результатам и т. д. Это заслуживает отдельного рассмотрения.

1.1. ЭВМ и управление

Управление — способ достижения в некотором объекте определенных целей. Объекты управления имеют различную природу. Ими могут быть и процесс решения математической задачи, и здоровье человека, и двигательный механизм любой степени сложности, и промышленное предприятие или экономика страны.

Обратимся к примеру. Пусть цель — поддержание постоянного уровня в резервуаре на пути потока жидкости (рис. 1.2). К такой цели часто сводится задача поддержания постоянной скорости вытекания.

Поведение указанного объекта с течением времени t можно описать соотношением

$$\frac{\Delta y}{S} = u - x, \qquad (1.1)$$

где y — высота уровня, с помощью которой логично описывать состояние объекта;

u — входной поток, т. е. объем втекающей в единицу времени жидкости; изменяя его, можно управлять состоянием;

x — выходной поток, препятствующий достижению цели;

S — сечение резервуара.

Требуемая цель могла быть легко достигнута при точно известной величине x. Тогда из (1.1) с учетом цели ($\Delta y = 0$) непосредственно находится нужное управление: $u = x$, т. е. входной поток должен повторять все изменения выходного.

На практике не всегда можно предусмотреть все величины, препятствующие достижению цели, но о них можно судить косвенно, по состоянию объекта. Вместо измерения x проще следить за изменением уровня жидкости, а принцип управления может быть очень простым: при понижении уровня приток жидкости нужно увеличить, и наоборот. Такая «противодействующая» зависимость управляющего воздействия от состояния объекта—один из примеров *отрицательной обратной связи.*

Формально такую связь можно описать как,

$$u = -kx, \qquad (1.2)$$

где k — фактор обратной связи, определяющий степень влияния состояния объекта на управляющее воздействие. Соотношение (1.2) означает, что понижение уровня жидкости должно сопровождаться пропорциональным увеличением ее притока. Если выходной поток скачкообразно изменяется от значения x_1 до x_2 (рис. 1.3), то без обратной связи уровень жидкости будет равномерно понижаться, а при ее наличии понижение уровня стремится к некоторой постоянной величине Δy_∞, называемой статической ошибкой регулирования. Эта ошибка обратно пропорциональна фактору обратной связи. При достаточно больших его значениях она может оказаться пренебрежимо малой, что и позволяет говорить о достижении цели.

Еще в 1765 году выдающимся русским изобретателем

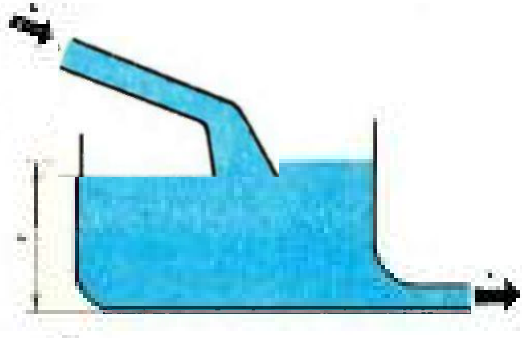


Рис. 1.2. Для простейшего описания состояния проточного резервуара достаточно учесть потоки втекающей (u) и вытекающей (x) жидкости, которые определяют положение уровня (y).

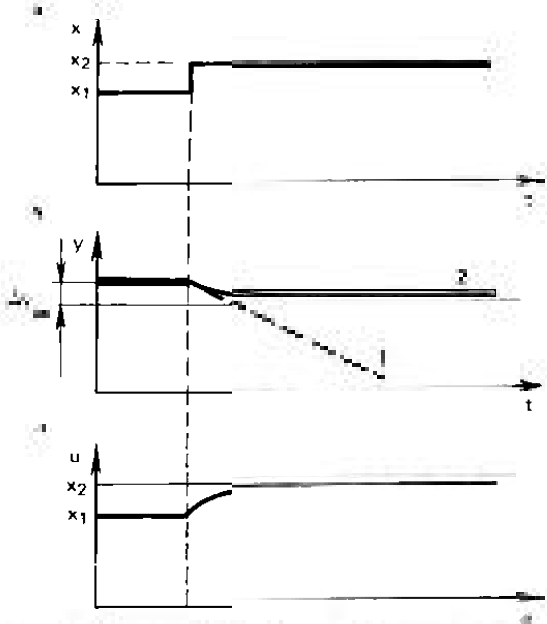


Рис. 1.3. Скачкообразное увеличение выходного потока *(а)* принципиально различным образом отражается на характере изменения уровня жидкости *(б)* в системах без отрицательной обратной связи (1) и при ее наличии (2). Благодаря действию обратной связи входной поток стремится компенсировать изменение выходного, но это происходит с запаздыванием *(в)*.

И. И. Ползуновым была предложена поплавковая система управления уровнем, которая с успехом применяется по сей день (рис. 1.4). Хорошо известны и другие регуляторы, основанные на отрицательной обратной связи, например центробежный регулятор скорости вращения Дж. Уатта. Управляющее воздействие формируется в них за счет энергии самой управляемой системы.

Однако источником энергии для управления не обязательно должен быть сам объект. Для управления уровнем в больших промышленных резервуарах более эффективна система с мощным

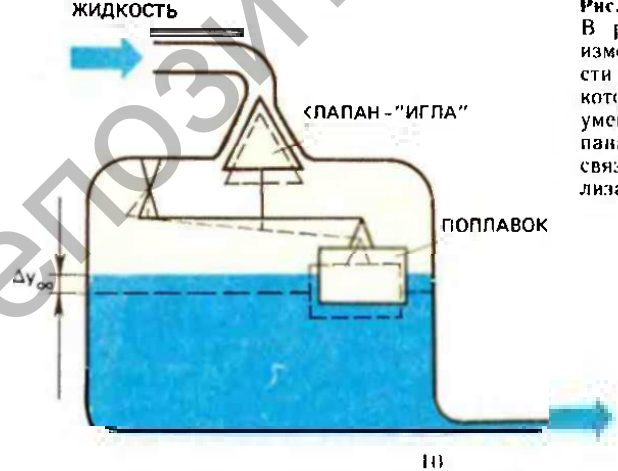


Рис. 1.4. В регуляторе Ползунова изменения уровня жидкости передаются поплавку, который увеличивает или уменьшает сечение клапана. Такая обратная связь приводит к стабилизации уровня.

насосом, включаемым электромагнитным реле. Сигналы для реле вырабатывает датчик уровня. И хотя они передают определенную энергию, правильная работа системы определяется, скорее, не энергией сигналов, а их «содержанием», т. е. информацией об объекте.

Основная функция любого устройства управления — передача и обработка информации.

Системами управления на основе понятия информации занимается *кибернетика* (от греч. *kybernetike* — рулевой) — наука об управлении.

Все величины, посредством которых может осуществляться взаимодействие объекта управления с окружающей средой, носят название связей. Обозначив информацию о совокупности связей, оказывающих воздействие среды на объект, через X, а объекта на среду — через Y, можно считать, что Y описывает текущее состояние объекта и

$$Y = F(X), \tag{1.3}$$

где F — преобразование объектом информации о воздействиях среды (рис. 1.5).

Чтобы управлять, нужно сформулировать цель управления, т. е. определить, к какому состоянию объекта следует стремиться, а из всех связей X выделить те, которые могут эффективно способствовать достижению цели. Информацию о них обозначим через U. Теперь поведение объекта удобнее описывается соотношением

$$Y = F(X,U), \tag{1.4}$$

а задача управления сводится к поиску воздействия, приводящего к заданному целью состоянию, т. е. к решению «уравнения»

$$Y^* = F(X,U), \tag{1.5}$$

где Y^* — требуемое состояние объекта, U — неизвестная величина.

В общей схеме управления (рис. 1.6) синтез управляющего воздействия осуществляет *управляющее устройство* (УУ), реагирующее на изменения как в среде, так и в объекте.

Такой подход имеет существенные достоинства.

Во-первых, при наличии информации об изменениях, происходящих в окружающей среде, управление может быть эффективнее, чем в системе с отрицательной обратной связью.

Во-вторых, независимо от природы объектов управление ими может быть основано на решении чисто математических задач. Специфика объектов при этом отражается на устройствах, воспринимающих информацию — датчиках и оказывающих непосредственное воздействие на объект — исполнительных механизмах.

Функцию обработки информации в системах управления выполняют ЭВМ, а нередко и микро-ЭВМ. Диапазон применения таких систем — от бытовых устройств до промышленных станков, агрегатов, машин.

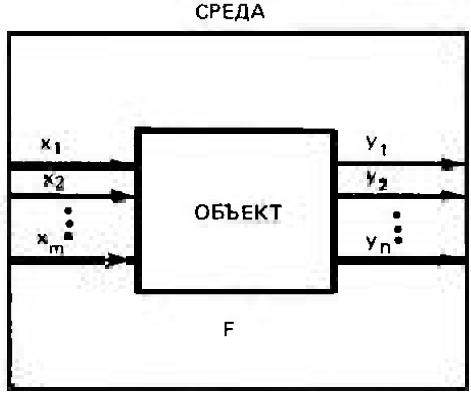


Рис. 1.5. С целью анализа объект управления обособляют от окружающей среды и изучают вскрытые в результате такого разделения связи. Во многих случаях они могут быть описаны конкретными, например физическими, величинами. Для простых объектов число связей может быть небольшим.

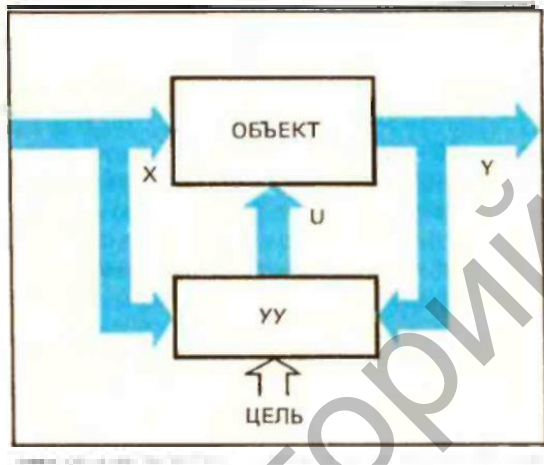


Рис. 1.6. В общей схеме управления, помимо объекта и окружающей среды, выделяется устройство управления (УУ), осуществляющее целенаправленное воздействие на объект.

Работа ЭВМ в системах управления формально сводится к решению уравнения (1.5), на основе которого формируются управляющие воздействия. Однако нередко число связей реального объекта настолько велико, что об уравнении и даже системе уравнений (1.5) можно говорить только условно. По этой причине более эффективным оказывается управление, осуществляемое человеком, который хотя и не решает никаких уравнений, но определенным образом увязывает решения со связями объекта. Этот процесс в значительной мере является искусством и опирается на интуицию и опыт.

Если же найдены способы формального описания преобразователя F объекта и из очень большого числа связей объекта со средой выделены важнейшие, имеет смысл говорить о модели объекта, представляющей несколько упрощенное его описание.

Опыт показывает, что всегда можно выбрать такое конечное число связей, при котором модель будет соответствовать объекту с необходимой точностью.

При очень сложных объектах построить подходящую модель помогают опять же ЭВМ.

Основой для машинного моделирования систем управления служат экспериментальные данные $\{X, U, Y\}$ о поведении объекта во всех исследованных к текущему моменту ситуациях. Эти данные обычно хранятся в памяти ЭВМ в виде т. н. *базы данных* (БД). Специальные программы — *системы управления базами данных* (СУБД) — обеспечивают быстрый доступ к любой хранящейся информации, ее пополнение и последующую обработку.

С помощью СУБД не составляет труда оперативно проверить, соответствует ли некоторая модель F_M данному объекту или нет, а значит, выбрать из всевозможных моделей наилучшую.

По ряду причин модель всегда бывает приближенной и нуждается в постоянном уточнении. Процесс подстройки модели к объекту называется адаптацией. Создание самоадаптирующихся систем управления — одно из перспективных применений ЭВМ.

1.5. ЭВМ и автоматизация производства

Рассмотрим этот вопрос на примере робота типа «механической руки», который осуществляет некоторые манипуляции с предметами на движущемся конвейере. Информация на электропривод робота поступает из устройства управления. При четком ритме работы конвейера оно должно всего лишь в строго определенные моменты времени включать или выключать нужные моторы и реле.

Работа таких роботов подкупает своей педантичностью и аккуратностью, они не устают, а развиваемые ими усилия и скорость работы могут во много раз превышать человеческие возможности.

Но вот в работе конвейера произошел маленький сбой: деталь оказалась не в нужном месте или на пути «руки» оказался некоторый предмет. Ситуация резко меняется: механизм, восхищавший нас своей эффективностью, выглядит теперь неуклюжим, смешным, а подчас и опасным,— ведь препятствием для его работы мог оказаться человек. Совершенно очевидно, что описанная ситуация — следствие несовершенства простейших устройств управления. Кроме того, будучи рассчитанными на выполнение определенной последовательности операций, они нуждаются в полной переделке при любых изменениях в технологическом процессе.

Создание более совершенных роботов неизбежно связано с использованием ЭВМ. Можно, например, предусмотреть возможность ручного управления работой робота и снабдить его датчиками, позволяющими ввести в ЭВМ информацию о всех произведенных манипуляциях. Запомнив выполненную человеком последовательность действий, ЭВМ в состоянии воспроизвести ее и повторить нужное количество раз. Даже если «обучение» робота поручить самому квалифицированному и опытному рабочему, то ЭВМ в состоянии «поискать» и найти более рациональные действия: например, из соображений минимальных затрат энергии приводов.

Более того, введенную последовательность действий робот может воспроизвести в несколько раз быстрее, чем в процессе «урока».

Дальнейшие пути совершенствования роботов — снабжение их фотоэлектрическими и иными датчиками «зрения», «слуха», «осязания», делающими их чувствительными к соприкосновению с предметом и т. п.

Последние поколения роботов служат основой создания *гибких производственных систем* (ГПС). ГПС работают с минимальным участием человека. Их перестройка на выпуск новой продукции фактически сводится к перекомпоновке оборудования и созданию новых программ для управляющих ЭВМ. Перспективы внедрения роботов со встроенными микропроцессорами в самые различные отрасли промышленности исключительно велики и предполагают колоссальный экономический эффект.

В чем же суть этой так называемой «микропроцессорной революции»?

Напомним, что важнейший фактор прогресса — повышение производительности труда, что в свою очередь подразумевает создание все более производительных машин и механизмов. В процессе взаимодействия человека с орудиями труда нетрудно выделить две важнейшие функции человека: приведение орудий труда в движение и управление ими. Возможности человека и как субъекта, осуществляющего управление, и как источника энергии для орудий труда ограничены. По этой причине совершенствование орудий труда неизбежно привело к созданию и использованию *автоматов* (слово «автомат» в переводе с греческого означает «самодействующий»). Появление автоматов в производстве высвободило человека от необходимости приводить в движение орудия труда, так как они черпали энергию от других источников (энергия ветра, падающей воды, пара и т. д.). Однако самодействующими такие устройства можно назвать лишь условно, поскольку в большинстве случаев управление ими осуществлялось человеком. Создание автоматов, самодействующих в буквальном смысле слова, стало возможным благодаря успехам кибернетики, вскрывшей информационную природу управления и предложившей методы создания самоуправляемых систем. Именно поэтому слово «автомат» в современной трактовке означает устройство преобразования и передачи информации. Сказанное объясняет, почему можно рассматривать ЭВМ (или, как вы помните, универсальные устройства обработки информации) в качестве важнейшего средства повышения производительности труда в народном хозяйстве.

Автоматизация коренным образом изменяет характер труда, являясь одной из предпосылок для стирания грани между трудом физическим и умственным.

1.6. ЭВМ в технике. Испытания конструкций и моделирование

Предположим, что нужно испытать на прочность корпус самолета. Для этого на него наклеиваются тензодатчики — резисторы, сопротивление которых зависит от величины деформации. Их число достигает тысяч, и подсоединять к каждому отдельный измерительный прибор нерационально, поэтому группы тензорезисторов обслуживаются одним измерительным прибором. Поочередное их подключение осуществляет коммутатор. У каждого коммутатора помещается оператор. Он снимает показания группы датчиков, т. е. производит измерения их сопротивлений в отсутствие механической нагрузки. После этого устанавливается определенная нагрузка, составляющая, например, 10 % от расчетной, и снова снимаются показания датчиков. Аналогичные измерения продолжаются далее для все возрастающих нагрузок.

Слабый узел конструкции выйдет из строя первым. Допустим, что он сломался при нагрузке, составляющей 80 % от расчетной. Испытания прекращаются. Вычисляют напряжение в дефектном узле, который затем переделывается с целью усиления. После восстановления разрушенной части испытания продолжаются до тех пор, пока конструкция не окажется равнопрочной, т. е. разрушающейся сразу во многих узлах. Если при этом нагрузка ниже 100 % от расчетной, то требуется усиление конструкции. В случае, когда нагрузка оказалась выше 100 % от расчетной, конструкция слишком прочна, и, следовательно, массу ее можно уменьшить. Снова требуется пересчет, переработка и связанная с ними обработка многих тысяч показаний датчиков при новых испытаниях.

Эксперимент усложняется еще более, если принять во внимание, что сопротивление тензорезисторов зависит не только от деформации, но и от температуры. Это приводит к необходимости учета внешних условий опыта, что особенно важно при разработке сверхзвуковых самолетов и космических систем, для работы различных узлов которых характерен неодинаковый аэродинамический нагрев.

Эффективность подобного рода экспериментов может быть существенно повышена путем автоматизации измерений и обработки данных с помощью ЭВМ.

Считывание и обработка показаний датчиков, сравнение экспериментальных данных с расчетными, незамедлительное информирование об отклонениях, превышающих допустимые,— все это может быть поручено вычислительной машине. Она немедленно выявит нелинейность деформации, являющуюся признаком скорого разрушения, и эксперимент будет прерван до разрушения слабого узла. Его замена ускорится, так как не придется восстанавливать смежные узлы.

Но, пожалуй, самая заманчивая возможность — моделирование поведения конструкций.

Уточним термин «модель» применительно к этой ситуации. Для любого реального явления характерно участие множества

взаимосвязанных процессов. Некоторые черты явления представляются важными, другие — несущественными. При изучении таких явлений неизбежен процесс упрощения, схематизации, заключающийся в выделении существенных факторов и пренебрежении остальными. В итоге получается *математическая модель явления*, изучать которую проще. Так, в физике возникли модели математического маятника, блоков без трения, невязких жидкостей и т. д. Прогнозировать их поведение в тех или иных условиях можно, решая соответствующие уравнения. Но несущественные на первый взгляд факторы оказываются важными в каких-то иных ситуациях. Набравшись терпения, мы заметим, что регулярные колебания маятника в конце концов прекращаются вопреки модельным представлениям о математическом маятнике и впечатлению, которое могло сложиться в результате краткосрочного наблюдения за обычным маятником. Подобным образом любая модель может оказаться неадекватной реальному поведению конструкции в какой-либо непредвиденной ситуации. Конечно, решающее слово всегда остается за экспериментом. Но ведь многие данные могут быть получены и при «чисто машинном» моделировании. Такие данные помогают адаптировать модель, т. е. сделать ее более точной и без проведения трудоемких и дорогостоящих экспериментов.

1.7. САПР

Моделирование — одна из характерных черт современных *систем автоматизированного проектирования* (САПР).

На базе наиболее совершенных ЭВМ создаются САПР для проектирования даже самих ЭВМ. Сообщения об этом журналисты снабжают сенсационными заголовками типа: «Суперкомпьютер превосходит самого себя, проектируя своего преемника», «Микросхемная реализация процессора — триумф автоматического проектирования» и т. п.

В действительности же дело заключается в том, что создание совершенной вычислительной техники сопряжено с большими объемами обрабатываемой информации, и без помощи САПР оно малоэффективно или же вообще невозможно.

Рассмотрим, например, не самый творческий, но очень ответственный и трудоемкий этап поиска оптимального расположения элементов интегральной микросхемы (о том, что это за элементы, читатель узнает в главе 5). Элементы должны быть размещены на небольшом кристалле (подложке) и соединены друг с другом тонкими металлическими полосками. Число плоскостей, в которых можно формировать межсоединения, невелико — в противном случае резко возрастает сложность производства микросхем. Рисунок межсоединений в каждой плоскости не должен содержать пересечений, иначе в схеме будут короткие замыкания. Поиск подходящего рисунка, называемый трассировкой соединений, напоминает разработку печатных плат для электронных приборов (такие платы есть, например, в телевизорах, приемниках, их широко

используют и радиолюбители в своих конструкциях). И в том, и в другом случае задача решается обычным методом проб и ошибок: если для выбранного расположения не удалось найти рисунка без пересечений, то берется новое расположение и т. д. В итоге на разработку даже не очень сложной печатной платы может понадобиться много часов работы.

А теперь представьте себе задачу трассировки межсоединений для современного микропроцессора, содержащего сотни тысяч элементов на подложке площадью меньше 1 см^2. Эта задача несоизмеримо сложнее, так как быстродействие современных электронных устройств нередко определяется рациональностью выбора путей передачи электрических сигналов. Несмотря на то что сигналы распространяются со скоростью, близкой к скорости света, их неуловимые по обычным представлениям задержки могут свести на нет быстродействие самих электронных элементов, поэтому процесс трассировки обязательно сопровождается моделированием задержки распространения сигналов. Если задержка в каком-то узле превышает допустимую, трассировка осуществляется заново. Наконец, когда модель оказывается пригодной, начинается изготовление самого электронного устройства.

И в этом, и в других случаях говорить об интеллектуальных возможностях ЭВМ, по крайней мере, преждевременно. Умственная деятельность конструктора в большинстве случаев не поддается переводу на язык программ, особенно в области выработки решений в сложных ситуациях. Машина просто перебирает все возможные варианты и может затратить на это занятие уйму времени, а квалифицированный специалист, если и не сразу примет верное решение, то, по крайней мере, укажет направление, существенно сужающее область поиска.

ЭВМ просчитает указанные варианты, а человек выберет из них оптимальный! Сочетание таланта и интуиции человека с точностью и быстродействием ЭВМ позволяет видеть в ней надежного помощника даже в процессе творчества.

Хотя ЭВМ берет на себя лишь нетворческую часть труда, ее можно рассматривать как усилитель умственных возможностей человека, подобно тому как, например, велосипед усиливает его физические возможности к передвижению.

Одна из особенностей взаимоотношений человека и ЭВМ в САПР — их *диалоговый*, или *интерактивный* (основанный на взаимодействии) характер. Посредником между человеком и машиной выступает терминал (в простейшем случае — электрическая пишущая машинка).

Современные *терминалы* — дисплеи сохранили от пишущей машинки лишь клавиатуру и обеспечивают отображение информации не на бумаге, а на экране *электронно-лучевой трубки* (ЭЛТ). В ходе диалога ЭВМ может информировать человека о заложенных в программу возможностях, предлагать свои варианты решения конкретных задач. Человек же, «печатая» на терминале специальные команды, заставляет ЭВМ выполнить ту или

иную функцию ее программы. Некоторые современные дисплеи позволяют осуществлять обмен и графической информацией. Так, конструктор может выбрать из БД интересующий его чертеж, схему или «нарисовать» их на экране дисплея с помощью «светового карандаша». Машина воспринимает эту информацию и в состоянии произвести ее обработку, например выровнять линии, нарисованные «от руки», внести в изображение изменения, придать ему более компактную форму, произвести процесс моделирования в заданных человеком условиях и т. п. Полученное на дисплее изображение может быть передано из памяти дисплея в память ЭВМ с целью обработки или размещения в БД. С последним обстоятельством связана другая важная особенность современных систем «человек — машина» — возможность *безбумажного хранения* и *обработки информации*. Еще недавно значительную часть рабочего времени конструкторов занимало оформление и переработка всевозможных чертежей, схем, таблиц и прочей документации. Малейшее изменение в ходе разработки зачастую приводило к необходимости полной переделки всей документации. САПР и СУБД позволяют осуществлять все изменения в памяти машины. При этом окончательный вывод исправленной или модифицированной документации производится с помощью управляемых от ЭВМ печатающих устройств, или графопостроителей. Сам вывод осуществляется намного быстрей и с более высоким качеством, чем у самого опытного чертежника.

1.8. Автоматизация работы служащих

По образному выражению социологов, основная задача служащих — предоставить в нужное время нужную информацию.

Кто же относится к категории тех, чей труд связан с поиском и обработкой информации? Прежде всего, это работники сферы управления, научных исследований, образования, проектирования, распределения информации, торговли, в какой-то степени деятели искусства и здравоохранения. Труд их преимущественно умственный, но далеко не во всех отношениях творческий. Он, как и любая умственная деятельность человека, содержит рутинную часть, т. е. работу, представляющую собой, в принципе, повторение одних и тех же действий, которые можно формально описать, а это значит поручить их выполнение ЭВМ. В таком смысле ЭВМ становится мощным средством интенсификации, т. е. повышения производительности умственного труда человека.

Средоточием информации в работе служащих традиционно является документ, а сама работа обычно складывается из поиска исходных документов, их анализа, переработки, представления результатов и общения с другими людьми. Рассмотрим, какими путями может повышаться эффективность такой работы.

Ускорению поиска данных из больших массивов призваны служить *информационно-поисковые системы* (ИПС). При условии, что вся представляющая интерес документация организована в

виде базы данных в памяти ЭВМ, специальные программы способны по запросу с терминала найти нужную информацию и представить ее в удобной форме несоизмеримо быстрее, чем человек. В этом убеждают примеры действующих ИПС: сравните, например, процесс получения авиабилета в кассах обычного типа и автоматизированной системы «Сирена». Не менее эффективно применение ИПС и во многих других сферах: архивных, справочных, диспетчерских, регистрационных и т. п. службах — там, где качество работы оценивается, прежде всего, умением быстро отыскать необходимую информацию.

Распространение ИПС сдерживается трудоемкостью переноса информации с бумажных носителей на машинные (магнитные диски и ленты внешних ЗУ ЭВМ). Это обычно делается «вручную», с клавиатуры терминала. Не случайно одна из целей разработки ЭВМ нового (пятого) поколения — наделение машин способностью воспринимать информацию в естественной для человека форме, т. е. в виде речи, документов, изображений.

Экспертные системы (ЭС), которые можно считать развитием ИПС, характеризуются бóльшими «интеллектуальными» способностями. Хранимая в памяти таких систем информация значительно полнее отражает совокупность знаний в конкретной области, чем база данных в ИПС. Это не просто совокупность фактических, предметных данных об интересующем классе объектов, но и элементы понятийного (концептуального) знания, включая связи, отношения и зависимости между объектами, реализованные в виде программ методы и приемы специалистов в данной области, а также алгоритмы решения типичных задач. Перечисленное образует так называемую *базу знаний* ЭС. Получив конкретный запрос, ЭС «включает» в работу все имеющиеся в базе знаний ресурсы и, если не дает окончательный ответ, то формулирует «интеллектуальный» совет, поясняя при необходимости, каким путем она пришла к тому или иному заключению.

Как и в случае ИПС, создание ЭС начинается с ввода данных в ЭВМ. Но это еще сложнее: одно дело — быть хорошим специалистом-экспертом, а другое — изложить свои знания в формализованном виде, приемлемом для ЭВМ. Передача ЭВМ человеческих знаний — одна из центральных проблем вычислительной техники, имеющих огромное практическое значение. Свидетельство тому — успешное применение ЭС в медицине, геологии, химии и других областях деятельности. Работы в области создания ЭС способствуют развитию наших представлений о характере мышления человека.

Еще одна распространенная форма работы, которую с успехом берут на себя ЭВМ,— *обработка текстов*. Между моментом, когда та или иная задача решена по существу, и представлением решения в виде, удобном для восприятия, как правило, скрывается обилие черновой работы. Сюда входит составление и, зачастую, многократная правка всевозможных документов: отчетов, таблиц, бланков, анкет, в т. ч. и тех, форма и содержание которых почти

не изменяются с течением времени. Эту работу значительно облегчают специальные программы-*редакторы текстов*. С их помощью процесс создания и редактирования текста происходит не на бумаге, а посредством терминала, в памяти ЭВМ.

Развитие функций таких программ — автоматическая нумерация страниц, разделов и параграфов, составление оглавления и списка литературы, расположение текста и заголовков в удобном для восприятия виде, соблюдение существующих стандартов на форму документов. На печать можно вывести только окончательную, выверенную и исправленную версию. Но в ряде случаев можно вообще обойтись без бумаги.

Небольшие приставки к телефонам, называемые *модемами*, с помощью специальных программ «электронной почты» позволяют практически мгновенно переслать документ в нужное место. Он автоматически появится на дисплее принимающей ЭВМ, а если в данный момент адресат отсутствует, то сообщение запишется в память, а машина напомнит о нем при первом же включении дисплея.

Только перечисленные возможности делают экономически оправданной установку ЭВМ для индивидуального пользования, или *персональных компьютеров* (ПК) на каждом рабочем месте. Работающие с ними «непрограммирующие профессионалы» — специалисты в конкретных областях деятельности — не обязаны иметь глубокой подготовки в области вычислительной техники. Особенностью ПК является «дружественный» характер реакции машины, точнее, заложенных в ней программ, на любые действия человека.

Со времени появления ПК прошло около десяти лет, но уже сейчас их влияние на общественный прогресс соизмеряют с последствиями развития книгопечатания, с той разницей, что ПК — не просто хранитель знаний, но и средство их активного использования в повседневной жизни.

* * *

Итак, практически в любой области на первый план выходят проблемы получения, хранения и обработки все возрастающих потоков информации, а ЭВМ нужны как инструмент усиления возможностей человека при обработке информации, как важнейшее средство повышения производительности труда в самых различных сферах.

Небезынтересно заметить, что название ранних этапов развития человечества было связано с названием материалов для орудий труда: каменный век, железный век и т. д. Далее отразились виды энергии, приводящей в движение орудия труда: век мускульной энергии, энергии ветра, пара, электричества и т. д. Сейчас нередко употребляют термин компьютерный век, подчеркивая тем самым то обстоятельство, что в наше время определяющее значение имеют способы обработки информации.

Глава 2

2.1. Что такое информация? Представление информации. 2.3. Цифровой автомат. 2.4. Как работу ЦА? 2.5. Алгоритмы. 2.6. Алгоритмические. 2.7. Универсальный массовый автомат – прообраз ЭВМ. Программное управление в ЦА. Переход к ЭВМ или принцип памяти программы. 2.10. Способы адресации. 2.11. Структура универсальных ЦА

АВТОМАТ, КОТОРЫЙ УМЕЕТ ВСЕ, ИЛИ ОСНОВЫ ИНФОРМАТИКИ

Человек обрабатывает информацию, руководствуясь не только правилами, но и тем, что мы называем здравым смыслом, интуицией. Машина же действует строго по правилам и «руководствуется» алгоритмом.

Не спешите утверждать, что алгоритм — набор правил, которые точно определяют последовательность действий. Ни рецепты поваренных книг, ни правила уличного движения, ни требования этики алгоритмами не являются. Наставления типа «немного посолить» или «с учетом ширины проезжей части» не удовлетворяют требованиям точности. В этике еще сложнее.

Вплоть до середины нашего века строгое определение алгоритма не смогли дать даже великие математики, пока не выяснилось, что точным во всех отношениях может быть только поведение машин.

По этой причине название главы следует понимать так: ЭВМ умеет делать все то, что умеют делать другие автоматы.

2.1. Что такое информация?

Информация — это какие-либо сведения о чем-то, совокупность данных об определенном объекте.

Такое толкование обусловлено происхождением самого термина (от лат. *informatio* — разъяснение, осведомление). Огромная популярность слова информация в наши дни определяется той ролью, которую начинают играть в нашей жизни ЭВМ, но подлинный смысл понятия информации глубже, чем интуитивное пред-

ставление о нем. Научное определение информации позволило подойти к совершенно разноплановым, на первый взгляд, процессам с единой точки зрения; подобным образом самые различные явления в физике объединяет понятие энергии.

Согласно наиболее общим воззрениям, информация — продукт отражения системой окружающей среды, или же продукт самоотражения системы. Под системой понимается материальный объект, помещенный в материальную среду. Если в объекте происходят какие-либо изменения, то он становится источником информации либо об окружающей среде, либо о протекающих в нем процессах. Информация имеет вполне материальные *носители* типа физических полей, нервных импульсов и т. п. Более высокая форма отражения характерна для организмов, обладающих центральной нервной системой, в которых разнообразная информация может складываться в целостное восприятие.

Мышлению человека свойственна высшая форма отражения объективной действительности — сознание. Наше понимание сути явлений и объектов представляет собой высшую форму информации, которая может соответствовать и нематериальным, воображаемым объектам.

Передача информации присуща всем явлениям природы. Однако, являясь пассивным отражением в системах неживой природы, информация становится активной в кибернетических или самоуправляемых системах, к которым можно отнести как живые организмы, так и автоматы.

Как весьма сложную кибернетическую систему можно рассматривать и человеческий мозг.

2.2. Формы представления информации

Информация может быть представлена в *аналоговой* или *дискретной* форме.

Если, например, электрическое напряжение или ток изменяются по тому же закону, что и некоторая другая физическая величина, то их называют электрическими аналогами этой физической величины. Разработано немало преобразователей различных параметров в их электрические аналоги. Подавляющее большинство преобразуемых величин изменяется непрерывно, поэтому и они, и их электрические аналоги принимают, строго говоря, бесконечное множество значений. По этой причине слово «аналоговый» стали отождествлять со словом «непрерывный». Оно употребляется сейчас лишь в этом смысле.

При дискретном или *цифровом* представлении информации используемая в качестве ее носителя физическая величина принимает конечное множество значений. Их удобно обозначать какими-либо символами.

Сравните лестницу и наклонную плоскость. В первом случае имеется строго определенное количество фиксированных высот, равное числу ступенек. Все их можно пронумеровать. Наклонная

плоскость соответствует бесконечному количеству значений высоты.

Лестницу можно приблизить к наклонной плоскости с какой угодно точностью, увеличивая число ступенек.

Подобная ситуация наблюдается и в технике. Если определена допустимая ошибка измерения, то любая аналоговая величина может быть заменена конечным набором своих дискретных значений, а они, в свою очередь, закодированы символами какого-нибудь алфавита, например пронумерованы обычными десятичными цифрами.

Закодированную информацию можно представить двумя принципиально различными способами. В первом из них символы кода в виде сигналов передаются *последовательно* один за другим по единственной линии связи, во втором — более быстром — группа символов передается *параллельно* по нескольким линиям связи одновременно.

2.3. Цифровой автомат

Назовем *цифровым автоматом* (ЦА) всякую искусственную систему обработки информации, представленной в дискретной форме. Такое устройство имеет вход и выход. На них используется параллельное представление информации, а ЦА для обработки последовательной информации выступает как частный случай. Совокупность входных и выходных сигналов ЦА в данный момент времени называют соответственно *входным* и *выходным словом*. Работу любого ЦА можно трактовать как преобразование входных слов в выходные слова того же или другого алфавита (рис. 2.1). Под алфавитом понимают множество всех использованных символов. Входные и выходные слова могут изменяться только в определенные моменты времени, называемые тактовыми. Их удобно обозначать целыми неотрицательными числами.

$$t = 0, 1, 2, \ldots, i, \ldots .$$

Интервал времени между двумя соседними тактовыми моментами называется *тактом*. Длительность тактов может быть неодинаковой, но в большинстве случаев они равновелики и определяются генератором синхронизации.

На первый взгляд ЦА напоминает словарь, который каждому слову ставит в соответствие иноязычный эквивалент. А вот для того чтобы он напоминал переводчика, необходимо учесть, что смысл слова, как правило, находится в тесной связи с предшеству-

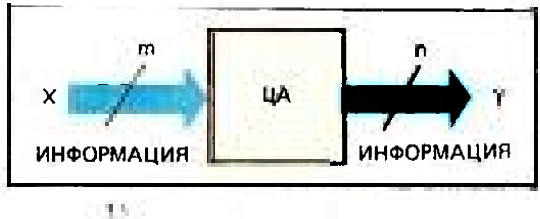


Рис. 2.1. Цифровой автомат можно рассматривать как преобразователь m-разрядных входных слов в n-разрядные выходные слова.

ющим текстом, поэтому в общем случае ЦА должен обладать способностью запоминать поступающую в него информацию. Однако в отличие от переводчика в некоторых случаях ЦА достаточно запомнить не само слово, а только факт его появления, что позволяет существенно экономить объем памяти.

Поясним сказанное примером. Типичный ЦА с памятью — обычные часы. Их показания в каждом такте зависят только от показаний в предыдущем такте. Сами показания являются выходными словами, а входной информацией служат сигналы с периодом в 1 секунду, вырабатываемые механическим или электрическим способом. За сутки на такой автомат поступает $24 \times 60 \times \times 60 = 86\,400$ входных слов и вырабатывается столько же различных выходных слов. Таким образом, часы имеют 86 400 состояний, но для их хранения, в принципе, достаточно пяти элементов памяти, каждый из которых способен запоминать лишь одну десятичную цифру. Обычно таких элементов шесть, и они соответствуют традиционному исчислению времени с помощью единиц и десятков часов, минут и секунд. При хранении же всех входных слов потребовалось бы 86 400 элементов для запоминания каждого входного сигнала.

Если же выходная информация зависит не только от факта появления, но и от значения многих предыдущих слов, объем памяти ЦА возрастает.

Так, цифровому автомату, предназначенному для выполнения арифметической операции сложения последовательно поступающих чисел, достаточно запомнить лишь одно слагаемое до прихода второго. А вот при обработке текста смысл предложения становится ясным только после приема последнего слова, поэтому вплоть до этого момента в памяти должны храниться, по крайней мере, все предыдущие слова предложения. Хранимую в памяти ЦА информацию о «предыстории» его работы принято называть *внутренним состоянием*. Удобно обозначать внутреннее состояние одним словом Q подходящей длины.

2.1. Как описать работу ЦА

Итак, ЦА под действием входного слова переходит из одного состояния в другое, сохраняя принятое состояние, по крайней мере, в течение такта, причем в общем случае выходное слово зависит не только от входного сигнала, но и от состояния автомата.

Любой цифровой автомат может быть построен из устройств двух типов — *комбинационных схем* (КС) и *схем с памятью*, которые сами могут рассматриваться как элементарные ЦА. Выходное слово КС в каждом такте однозначно определяется входным словом, и работу КС можно формально описать функцией

$$Y = \Phi(X).$$

Простейший способ задания $\Phi(X)$ – таблица, в которой всем возможным входным словам X ставятся в соответствие значения Y, например таблица 2.1.

Таблица 2.1

Входное слово X		Выходное слово Y	
x_1	x_2	y_1	y_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Схемы с памятью, а в общем случае и произвольные ЦА описываются выражениями

$$Y_{t+1} = \Phi(X_t, Q_t),$$
$$Q_{t+1} = F(X_t, Q_t),$$

где индексы $t+1$ и t относятся, соответственно, к новому и предыдущему тактам работы ЦА. Функции Φ и F и в этом случае удобно задавать таблично. Совместное изображение функций Φ и F получило название *таблицы переходов* ЦА. В такой таблице, например таблице 2.2, предыдущее состояние ЦА выступает на правах аргу-

Таблица 2.2

Предыдущее состояние ЦА Q_t	Входное слово X		Выходное слово Y	Новое состояние Q_{t+1}
$q_1(t)$	$x_1(t)$	$x_2(t)$	$y_1(t+1)$	$q_1(t+1)$
0	0	0	0	1
0	0	1	0	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	0	1
1	1	0	1	1
1	1	1	0	0

мента, а новое — в качестве функции. В простейших схемах с памятью, которые будут рассматриваться в главе 6, внутреннее состояние обычно отождествляют с выходным словом, а их работа описывается одной функцией

$$\Phi \equiv F.$$

Структура и работа любого ЦА тесно связаны с формой представления информации. Если вся входная информация может быть представлена параллельно, в виде одного слова, то запоминание ее необходимо лишь в тех случаях, когда элементы ЦА не рассчитаны

на обработку длинных слов. В остальных случаях весь ЦА может быть выполнен в виде одной КС, и его быстродействие будет максимальным.

При последовательной форме представления информации неизбежно ее запоминание. По этой причине схемы с памятью иногда называют последовательными.

На практике обычно принимается некоторое компромиссное решение: информация представляется в смешанной параллельно-последовательной форме, чем обеспечивается приемлемое соотношение сложности устройства ЦА и его быстродействия.

2.5. Алгоритм и интуиция

Еще раз отметим различный характер обработки информации в ЦА типа комбинационных схем (КС) и в ЦА с памятью.

В комбинационных схемах выходное слово формируется из входного со скоростью распространения сигнала через схему. Происходит это в одном такте.

В автоматах с памятью имеет место процесс обработки информации, распадающийся на звенья, каждое из которых выполняется в течение одного такта и может быть описано на языке КС. С этим процессом и связано понятием *алгоритма* обработки информации. (Слово «алгоритм» произошло от имени знаменитого математика IX века Аль-Хорезми, сформулировавшего правила арифметических действий над числами.)

В математике получило широкое распространение определение алгоритма как точного предписания конечной последовательности однозначных, понятных действий, направленных на достижение указанной цели, исходя из некоторых начальных данных.

Уточним особенности сформулированного определения:

1. Алгоритмы создаются для исполнителей, которых интересует конечный результат, а не тонкости процесса решения задачи. Даже когда алгоритм создается «для себя», то преследуется цель ускорить и упростить решение, так чтобы это можно было сделать не задумываясь, «автоматически».

2. Отдельный алгоритм зачастую составляет часть другого, более сложного алгоритма. Так, алгоритм решения квадратного уравнения опирается на вспомогательные алгоритмы алгебраических операций над числами. Вообще, решение любой сложной задачи существенно упрощается, если ее можно свести к последовательности подзадач. Это тем более имеет смысл, если подзадачи типичны для широкого круга других задач.

3. Любой алгоритм является дискретным, т. е. распадается на последовательность предписаний, направленных на выполнение некоторых элементарных действий или операций. Эти предписания должны быть понятны исполнителю. Образно говоря, исполнитель должен хорошо владеть языком, на котором описан алгоритм, в противном случае он нуждается в услугах хорошего переводчика.

4. Исполнитель должен уметь правильно производить все элементарные действия, содержащиеся в алгоритме.

5. Алгоритмам обычно свойственна массовость, т. е. применимость к различным исходным данным. Решение любой задачи «в общем виде» по существу всегда представляет собой некоторый алгоритм.

6. Наконец, алгоритм должен быть результативным, т. е. приводить к цели за конечное число шагов. Если же процесс решения завершается безрезультатно или не заканчивается вовсе, то говорят, что алгоритм не применим к взятым исходным данным.

Несмотря на значительные достижения в разработке и распространении всевозможных алгоритмов в математике, попытки научного подхода к алгоритмам вплоть до XX века были малоуспешными. Причина — трудоемкость строгого, формального определения понятия алгоритм. Формулировка, приведенная выше, может быть названа определением лишь в интуитивном смысле. Она не является строгой. В ней нет указаний на то, что может быть объектами алгоритма, а понятия типа «точное предписание», «понятные действия» — расплывчаты.

2.6. Алгоритм и программа

Отсутствие формального определения алгоритма приводило к ряду заблуждений. Так, немецкий математик Готфрид Вильгельм Лейбниц (1646—1716) пытался построить ни более ни менее как общий алгоритм решения любой алгебраической задачи. Аналогичные попытки в несколько более конкретной форме предпринимались даже в начале нашего века. Безуспешность всех этих поисков постепенно привела к выводу о том, что такие задачи *алгоритмически неразрешимы*. Проблема формального определения алгоритма стала еще более острой. Ее решение было найдено лишь в 30-х годах нашего столетия. В его основе лежало осознание того, что любой алгоритм оперирует не с реальными объектами, а с их символическими отображениями. Несколько позднее эту мысль удалось выразить еще более точно: объектами алгоритмов является информация, представленная в дискретной форме. Выяснилось также, что гарантией точности и однозначности алгоритма служит однозначность и точность языка, на котором он описывается. Причина расплывчатости интуитивного определения алгоритма заключается в том, что его исполнителем неявно предполагался человек, получающий предписания на естественном языке, одна из важнейших особенностей которого — многозначность. Она обусловлена наличием омонимов, синонимов, контекстностью, т. е. существенной зависимостью смысла от предшествующего изложения и т. п. Человеческому языку свойственна избыточность: пониманию текста зачастую не препятствуют как оговорки, описки, пропуски букв, так и сокращения целых слов. Все эти особенности способствуют образности, яркости и насыщенности при выраже-

нии самых различных мыслей, эмоций, но затрудняют процесс формализации многих понятий, в том числе и алгоритма.

Не исключено, что подобного рода соображения привели английского ученого А. Тьюринга в 1936 году к идее: предложить схему некоторой машины и назвать алгоритмами все то, что она умеет делать.

Не вдаваясь в детали устройства, отметим, что *машина Тьюринга* (МТ) представляет собой цифровой автомат, оперирующий словами единичной длины (т. е. символами) и снабженный запоминающим устройством в виде бесконечной ленты, на которой записываются символы входного слова, промежуточные результаты, а в итоге работы — и выходное слово.

Алфавит информационных слов, с которыми оперирует машина Тьюринга, может быть произвольным, но конечным; он называется внешним. Помимо него, необходим внутренний алфавит для обозначения внутренних состояний и некоторых особых ситуаций. В результате работа машины Тьюринга допускает символическое описание в виде специальной таблицы переходов. В каждой строке этой таблицы текущему входному символу и текущему состоянию ставится в соответствие выходной символ, новое состояние и указание, слева или справа от текущего нужно взять следующий обрабатываемый символ. Строки таблицы называются *командами*, а таблица в целом — *программой*. Работа МТ начинается с «настройки» на начальное внутреннее состояние и первый символ входного слова. По ним в программе находится нужная команда, а в результате — выходной символ, новое состояние и место, откуда нужно считать следующий входной символ. Так происходит до тех пор, пока не будет достигнуто особое, «конечное» состояние, соответствующее решению задачи.

Основной результат работы Тьюринга заключается не в том, что для любого алгоритма в интуитивном смысле может быть построена соответствующая машина, хотя этот факт и не вызывает сомнений в результате многочисленных проверок. С помощью работы Тьюринга можно доказать, что МТ не может решать задачи, которые интуитивно неразрешимы, в том числе упоминавшуюся проблему Г. Лейбница.

Неразрешимость таких задач на машине Тьюринга не есть следствие ее примитивности. Тьюринг, действительно, искал как можно более простую схему, позволяющую не только формализовать процесс решения, но и обладающую применимостью к любым задачам. Предположение Тьюринга о том, что всякий алгоритм может быть реализован соответствующей машиной, было всего лишь гипотезой. Однако весь последующий опыт позволил возвести этот тезис в ранг формального определения алгоритма. И пожалуй, самым впечатляющим доказательством справедливости идеи Тьюринга явилось установление эквивалентности этого определения другим формальным определениям, данным независимо советскими математиками А. А. Марковым и А. Н. Колмогоровым и американцем Э. Постом.

2.7. Универсальная машина Тьюринга — прообраз ЭВМ

Машина Тьюринга — воображаемая конструкция, построить ее — значит выбрать подходящий для данной задачи алфавит, написать символическую программу и убедиться, будет ли достигнуто в ходе ее выполнения конечное состояние. Это можно сделать путем рассуждений, даже не получив конечного результата.

Работа МТ происходит в нашем воображении. Следовательно, машины Тьюринга — вовсе не машины в обычном смысле, а скорее тексты на некотором языке, что вполне соответствует нашему интуитивному представлению об алгоритме.

А является ли алгоритмом работа любой МТ? В поведении МТ есть много общего: все, что они «умеют», сводится к достаточно простым операциям поиска текущей команды, преобразования входного символа в выходной, изменения внутреннего состояния и нахождения следующего входного символа. Эти операции носят настолько регулярный, «механический» характер, что их вполне можно описать в виде единственной программы для подходящей машины — *универсальной машины Тьюринга* (УМТ).

Идея УМТ заключается в имитации работы любой конкретной МТ. С этой целью в памяти УМТ должны храниться программа работы и образ имитируемой машины, включающий информацию как о содержимом ее памяти с указанием на текущий входной символ, так и о текущем состоянии.

На первый взгляд это кажется невозможным, ведь множество всех имитируемых машин бесконечно. Но тексты на любых языках можно зашифровать (закодировать) с помощью единственного, например цифрового алфавита. Сделать это, очевидно, нужно так, чтобы можно было достаточно просто различать программу от образа машины, символы от состояний и т. п. В этом случае программа работы УМТ сводится к выполнению системы несложных действий.

В образе МТ ищутся коды первого символа и начального состояния. По ним в зашифрованном тексте программы находится нужная команда, несущая информацию о тех изменениях, которые нужно произвести в образе МТ. После их совершения коды нового состояния и следующего символа позволяют отыскать следующую команду и т. д. Этот процесс, называемый *интерпретацией*, продолжается до тех пор, пока не будет достигнуто конечное состояние. В этом случае в памяти УМТ окажется код выходного слова имитируемой МТ.

В воображаемом устройстве УМТ использованы важные идеи, нашедшие позднее воплощение в особенностях конструкции и работы настоящих ЭВМ. В первую очередь, это работа по программе, представляющей собой символическую запись алгоритма на языке, «понятном» для исполнения не очень сложным устройством. Во-вторых, это наличие запоминающего устройства для хранения как исходной, промежуточной и конечной информации, так и самой программы. Память реальных ЭВМ всегда конечна, но это ограни-

чение становится несущественным в современных ЭВМ, обладающих весьма большой памятью. Наконец, это возможность имитации работы любых специализированных устройств обработки информации на одной универсальной конструкции.

2.8. Программное управление в ЦА

Итак, мы вправе считать ЦА устройством, реализующим конкретный алгоритм обработки информации. Его основная деятельность — операции над символами — выполняется комбинационными схемами. Но в нем происходят и другие процессы типа изменения состояния схем с памятью, записи слов в запоминающее устройство. Все они должны быть «привязаны» к определенным тактам, наконец, должно иметься устройство фиксации самих тактовых моментов.

Эта достаточно сложная роль в модели Тьюринга поручена человеку, а в реальных ЦА — устройству управления. В результате любой ЦА можно представить состоящим из *запоминающего устройства* (ЗУ), *операционного устройства* (ОУ) и *устройства управления* (УУ) (рис. 2.2).

В состав ОУ входят комбинационные схемы, общая функция которых — трансформация произвольного символа выбранного алфавита в любой другой символ этого алфавита. В главе 4 будет показано, как это можно сделать с помощью конечного набора элементарных операций. Всякое действие, инициируемое одним управляющим сигналом и выполняемое за один такт, называется *командой*. Таким образом, функция УУ — порождение соответ-

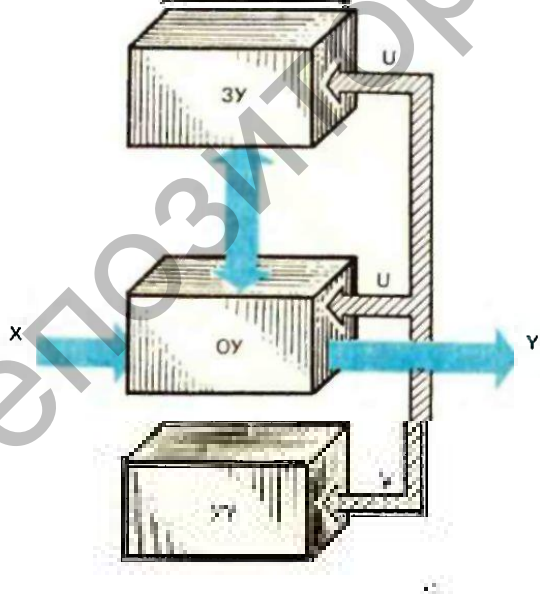


Рис. 2.2
В процессе работы цифрового автомата ОУ производит обмен информацией с ЗУ и окружающей средой. Выделены сигналы управления, поступающие из УУ. В ответ на них остальные блоки могут выдавать сигналы оповещения *V*, сигнализирующие УУ об окончании выполнения текущей операции, особых ситуациях в ходе ее выполнения и т. п.

Рис. 2.3.
ЦА, использующие принцип программного управления, не могут работать без символической (закодированной) программы, записанной в явном виде на каком-либо носителе информации.

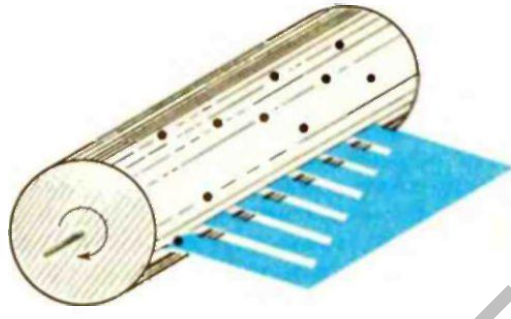


Рис. 2.4.
Работа несложного музыкального автомата — одно из самых древних воплощений принципа программного управления. Штырьки, расположенные на вращающемся барабане, задевают при своем движении вибрирующие пластинки, издающие звуки различной высоты. Продуманное расположение штырьков позволяет воспроизвести некоторую мелодию. Чтобы заставить автомат исполнить новую мелодию, достаточно расположить штырьки иначе.

ствующей выбранному алгоритму последовательности команд, т. е. *программы*.

До сих пор мы говорили о программе как способе символического описания алгоритма. Теперь понятие программы приобретает смысл способа функционирования ЦА, действия которого инициируются последовательностью сигналов УУ. Если устройство управления рассчитано на выполнение единственной программы и работает без внешних воздействий, то говорят, что оно обладает *жесткой или схемной логикой* работы.

При другом подходе УУ выдает управляющие сигналы лишь тогда, когда на его специальном входе присутствует код текущей команды программы. Выполнение каждой команды разбивается на считывание этого кода с внешнего носителя информации и собственно исполнение.

Само УУ играет роль преобразователя закодированных команд программы в управляющие сигналы. Такой подход к организации УУ называется *принципом программного управления* (рис. 2.3). Он позволяет перейти к реализации другого алгоритма без каких бы то ни было переделок УУ. Для этого достаточно записать на носитель информации новую программу работы.

Принцип программного управления известен человечеству давно и реализован в музыкальных автоматах (рис. 2.4), игрушках, многих механизмах. В качестве носителя информации в таких устройствах использовались вращающиеся диски, барабаны с отверстиями или выступами, зубчатые колеса, перфоленты. К сожалению, все они имеют низкую скорость работы.

Воображаемая машина Тьюринга подсказывает один из способов построения запоминающих устройств (ЗУ). Длинная лента с символами или отверстиями (перфолента) – не что иное, как *ЗУ с последовательным* доступом; найти нужный символ (или группу символов) на ней можно лишь путем последовательного просмотра всех предшествующих записей. Такие ЗУ удобны для хранения программы, потому что команды программы обычно выполняются последовательно.

Что же касается хранящихся в ЗУ символов, с которыми оперирует программа, то для них последовательный характер доступа — не самый удачный. В общем случае нужная цифровому автомату информация может находиться в произвольных местах (ячейках) ЗУ и возникает проблема, как ее разыскать. Эта проблема может решаться различными путями. Наиболее распространен подход, при котором все ячейки ЗУ нумеруются, а нужная из них разыскивается по номеру, называемому адресом. ЗУ, использующие этот принцип, называются *адресными*. Последовательные ЗУ также можно превратить в адресные, если на носителе рядом с самим символом хранить его адрес.

Последовательный характер работы ЗУ неминуемо снижает быстродействие. Действительно, если после символа, расположенного в одном «конце» ЗУ, понадобилась информация из другого «конца», необходим бесполезный просмотр содержимого всех промежуточных ячеек.

Адресные ЗУ с одинаковым временем доступа ко всем ячейкам получили название *ЗУ с произвольной выборкой* (ЗУПВ). Время доступа к ячейке с любым адресом в наиболее совершенных из них составляет десятые и даже сотые доли микросекунды! Быстродействие же операционных устройств, основу которых составляют комбинационные схемы, может быть еще выше.

А что, если сама программа будет поступать в УУ не с внешнего носителя, а из ЗУПВ?

Принято считать, что эта идея принадлежит американцу венгерского происхождения Дж. фон Нейману, принявшему весьма активное участие в становлении и развитии вычислительной техники. Метод, при котором закодированная программа работы ЦА размещается вместе с обрабатываемой информацией в ЗУ ЦА, получил название *принципа хранимой в памяти программы*, а само ЗУПВ — оперативного запоминающего устройства (ОЗУ). Свое применение он нашел уже в ранних моделях ЭВМ и с тех пор является одной из основных, неотъемлемых особенностей любого компьютера.

Адресный характер ОЗУ приводит к тому, что команды, на которые теоретически распадается алгоритм решения любой задачи, в общем случае содержат следующую информацию:

1) адрес преобразуемого символа;

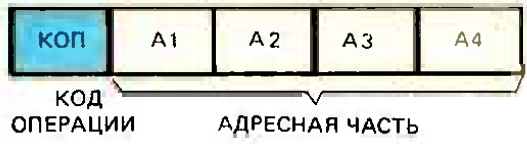


Рис. 2.5. Четырехадресная команда содержит код операции (КОП), т. е. символическое обозначение выполняемого действия, а также адреса первого и второго операндов, результата действия над ними и следующей команды (соответственно: A1, A2, A3, A4).

2) указание на элементарную операцию, которую нужно совершить над этим символом;
3) адрес ячейки ОЗУ, куда нужно поместить результат;
4) адрес ячейки ОЗУ, где хранится следующая команда программы.

Помимо этого, команда в общем случае должна учитывать внутреннее состояние УУ и при необходимости изменять его.

На практике, кроме операций трансформации одного символа в другой, удобны операции над парами символов. Рассмотрим, например, как выполняет МТ сложение двух чисел, допустим 3 и 4. В первом такте она «видит» цифру 3. Чтобы ее запомнить, есть только один выход — изменить свое состояние. Когда на следующем такте МТ «видит» четверку и пишет вместо нее 7, то это можно трактовать как трансформацию символа 4 в 7 с учетом предыдущего состояния. С таким же успехом можно считать, что символ 7 получается в результате действия (сложения) сразу над двумя символами 3 и 4, а внутреннее состояние не играет роли.

Следовательно, в реальных автоматах могут быть очень удобны команды, оперирующие сразу двумя символами. Будем называть символы, а в общем случае — любые порции информации, с которыми оперирует команда, *операндами*. (Если ЦА использует параллельное представление информации, то операндом может быть и слово.) Тогда любая команда в общем случае должна состоять из составляющих, представленных на рисунке 2.5. Команды такого вида называют четырехадресными. Они могут занимать в ОЗУ сразу несколько последовательных ячеек. Конечно, УУ должно уметь «распознать», сколько ячеек занимает считываемая команда, чтобы случайно не принять в качестве продолжения этой команды начало следующей или посторонний операнд. С этой целью код операции (КОП) всегда располагается в первой из ячеек, занятых командой, и обязательно содержит информацию об их суммарном числе. При этом исключается всякая путаница, ведь A4 указывает на первую ячейку следующей команды с ее КОП, который «подсказывает» УУ, сколько ячеек нужно «прочитать» еще.

Простота составления программы с помощью четырехадресных команд имеет и негативную сторону. Длина команд получается большой, программа занимает много места в ОЗУ, снижается скорость ее выполнения. Существует несколько путей преодоления этих сложностей, называемых проблемой адресации. Рассмотрим некоторые из них.

Если в ЗУ с последовательной выборкой расположить команды в той последовательности, в какой они следуют по программе, то переход от одной команды к следующей будет происходить автоматически, и в адресе А4 нет необходимости.

Для ЗУПВ можно исключить из каждой команды адрес А4, если ввести в состав УУ специальное устройство — *счетчик команд* (СК). В начале работы этот счетчик содержит адрес первой команды программы. Распознав ее КОП, УУ увеличивает содержимое СК на число ячеек, занятых этой командой. Если команды в ОЗУ расположены последовательно, то новое содержимое СК теперь является адресом второй команды. Нетрудно заметить, что в общем случае СК служит указателем адреса команды, следующей за текущей. В результате все команды могут быть трехадресными.

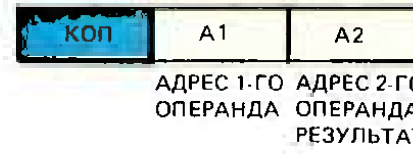
Заметим, что если нужно нарушить естественную последовательность выполнения команд, например из-за изменившегося в результате действия оповещающих сигналов состояния УУ, или при реализации разветвляющегося алгоритма, то для этого необходимо принудительно изменить содержимое СК. С этой целью используют специальные *команды передачи управления* (ПУ). Возможная их структура представлена на рисунке 2.6. *Команды условной передачи управления* удобно использовать для алгоритмов, содержащих ветвление в зависимости от выполнения или невыполнения какого-либо условия.

Команды безусловной передачи управления не содержат кода условия и всегда обеспечивают переход к команде, расположенной по адресу А. Они удобны, если в последовательном расположении команд программы в памяти есть «островки», содержащие операнды или побочные ветви программы.

Наличие команд передачи управления позволяет рассматривать содержимое счетчика команд как аналог текущего состояния МТ.

Рис. 2.6. Команда условной передачи управления содержит адрес команды, к выполнению которой должно перейти УУ, если выполнено некоторое условие, например если внутреннее состояние УУ соответствует нужному.

Рис. 2.7
В двухадресной команде адреса результата операции (А3) и следующей команды программы (А4) содержатся неявно: А3 = А2, а А4 определяется посредством счетчика команд.

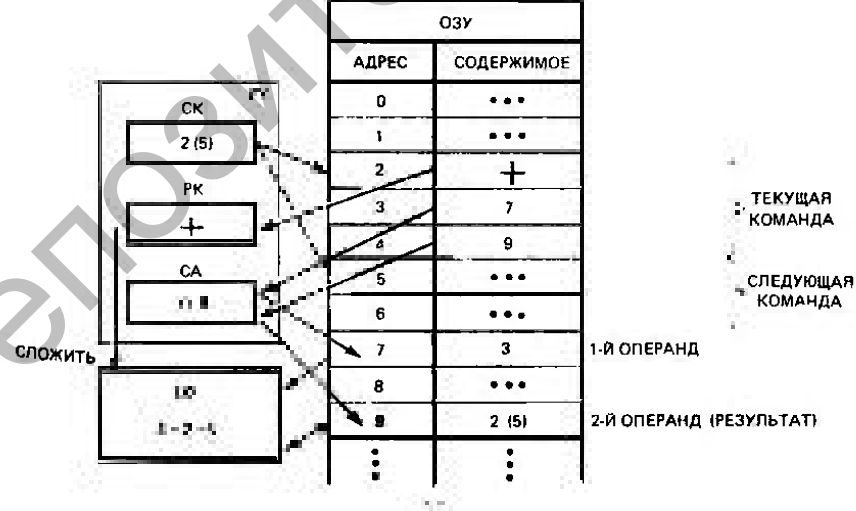


Путь дальнейшего сокращения адресной части команд — размещение в ОЗУ результата выполнения команды на месте одного из операндов, обычно второго. В результате команды ЦА могут быть сделаны двухадресными (рис. 2.7).

Процесс выполнения *двухадресной команды* рассмотрим на примере. Пусть набор символов, с которыми оперирует ЦА, включает десятичные цифры и нас интересует команда сложения чисел 3 и 2, расположенных в ОЗУ по адресам А1 = 7 и А2 = 9. Такая команда может быть закодирована символами +, 7, 9. Предположим, что каждая ячейка ОЗУ рассчитана на хранение одного символа, и символ + находится во второй ячейке. Тогда перед выполнением команды содержимое СК должно равняться 2. Для доступа к указанным в команде операндам УУ использует *систему адресации* (СА), обеспечивающую обращение к ячейкам ОЗУ по их адресам (рис. 2.8).

Еще большее сокращение адресной части команд допускает

Рис. 2.8.
В процессе выполнения «команды» + 7,9 код операции (+) поступает для хранения в специальное устройство — регистр команд (РК) и определяет характер действия, производимого в ОУ. По адресам А1 = 7 и А2 = 9 СА находит операнды (3 и 2), передает их в ОУ и после выполнения сложения помещает результат по адресу А2 = 9. В круглых скобках на рисунке обозначены состояния ячеек памяти и СК, изменившиеся в результате выполнения команды.

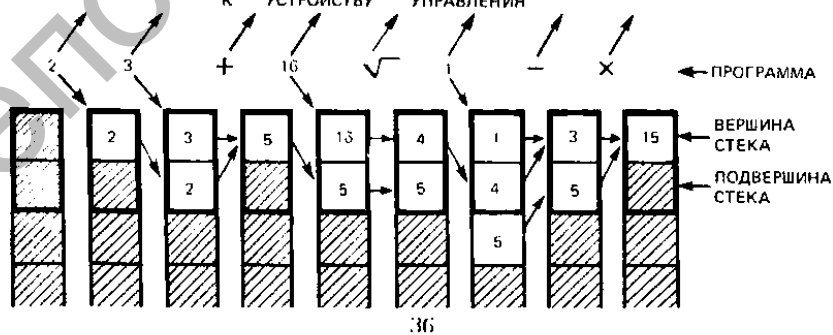


использование так называемого *сверхоперативного запоминающего устройства* (СОЗУ) — небольшого количества быстродействующих элементов памяти, объединенных в *регистры*, устройство которых будет пояснено в 6-й главе. Преимущества *регистровой памяти* обусловлены тем, что часто текущая команда программы использует в качестве одного и даже двух операндов результаты непосредственно предшествующих ей команд. Если они заносятся в СОЗУ, то адресная часть команд не будет занимать много места, т. к. адреса регистров СОЗУ в силу небольшого их числа весьма коротки. Такой метод адресации операндов называется *регистровым*. Если же вместо самих операндов в СОЗУ хранить их адреса A1 и A2, то появляется возможность доступа к операндам всего ОЗУ. Адресная часть команд при этом не увеличивается, так как в ней называются не сами A1 и A2, а короткие адреса регистров. Такая адресация называется *косвенно-регистровой*. С использованием регистров СОЗУ можно организовать и другие методы адресации, поэтому их называют *регистрами общего назначения* (РОН).

Большинство современных ЭВМ рассчитано на работу с двухадресными командами и широким использованием РОН. Как частный случай возможны *одноадресные команды* (например, передачи управления) и *безадресные команды*, не обращающиеся к ЗУ.

В принципе существует возможность построения ЦА, использующих только *нуль-адресные команды*, т. е. команды, не имеющие явного указания ни на один из операндов. Эта возможность основана на представлении алгоритмов в виде т. н. бесскобочной или *польской инверсной записи* (ПОЛИЗ). При этом исключительно удобна организация ЗУ для хранения операндов по принципу «последним пришел — первым вышел» или LIFO (от англ. *Last In-First Out*). ЗУ, использующее этот принцип, называют

Рис. 2.9. Воспринимая программу работы, УУ нульадресной машины должно различать операнды (2, 3, 16, 1) от знаков операций над ними ($+$, $\sqrt{\ }$, $-$, $\times$). Если очередная «команда» — операнд, то его нужно «протолкнуть» в стек; если это — знак операции, то нужно выполнить данную операцию. Говоря о «проталкивании» и «выталкивании», имеют в виду аналогию между работой стека и магазина для патронов в пистолете-автомате.



стеком: оно запоминает символы в порядке их поступления, а выдает в обратном.

Возможность использования нуль-адресных команд и работу стека поясним на примере вычисления арифметического выражения $(2+3)\times(\sqrt{16}-1)$. В ПОЛИЗ это выражение пишется 2,2, +, 16, $\sqrt{}$, 1, —, ×, т. е. так, чтобы знаки операций следовали за соответствующими операндами. Легко заметить, что при этом порядок действий определен совершенно однозначно, хотя скобки не используются. Последнее выражение и представляет собой программу работы УУ, выполнение которой требует, чтобы каждое арифметическое действие производилось над операндами, расположенными в вершине и подвершине стека. При этом результат помещается в вершину, а остальное содержимое стека «выталкивается» на одну позицию вверх. Для одноместных (с одним операндом) действий типа извлечения корня операция производится только над вершиной стека. На рисунке 2.9 изображено содержимое стека после выполнения каждой команды. В начале стек пуст.

Стековые вычисления широко применяются в микрокалькуляторах.

2.11. Что делает ЭВМ универсальным ЦА?

В словарях можно найти примерно следующее определение: ЭВМ — универсальное автоматическое устройство обработки цифровой информации, особенностями которого являются принцип программного управления и принцип хранимой в памяти программы. Очевидное отличие ЭВМ от любого другого ЦА заключается в *универсальности,* под которой, согласно теории алгоритмов, следует понимать возможность интерпретировать работу любого автомата, т. е. реализовать произвольный алгоритм.

Как достигается универсальность?

Подытожим известное.

1. Устройство должно допускать описание своей работы на языке машины Тьюринга. Этому требованию отвечает любой ЦА, в том числе и с жесткой логикой, состоящий из комбинационных схем и схем с памятью. Функции управления им способно осуществлять простейшее устройство, состоящее из *генератора тактов.*

2. Техническое воплощение в УУ принципа программного управления позволяет решать множество задач, для представления которых достаточно одного алфавита. Программа, соответствующая выбранному алгоритму, размещается на внешнем носителе и может быть легко изменена. Нужные изменения внутреннего состояния ЦА могут достигаться действием оповещающих сигналов на элементы памяти УУ. Последовательный во времени характер обработки во многих случаях приводит к использованию ЗУ для хранения входной информации и промежуточных результатов.

3. Реализация принципа хранимой в памяти программы повышает быстродействие УУ до быстродействия запоминающих

устройств. Важную роль при этом приобретает проблема адресации. Каждая хранящаяся в памяти команда программы состоит из кода производимого ею действия и может содержать адресную часть, указывающую УУ на место расположения нужных операндов в ЗУ. Изменения состояния УУ могут отражаться поведением счетчика команд. Возможность его принудительной установки с помощью команд передачи управления позволяет очень удобно учитывать изменения состояния ЦА в самой программе. При трансформации символов удобны двухоперандные (двухместные) операции.

4. При таком полном учете внутренних состояний автомата универсальность как способность решать все задачи, написанные на некотором фиксированном алфавите, достигается при условии, что существует набор команд, позволяющий осуществить произвольное преобразование типа «операнд-операнд» или «пара операндов-операнд». Будем называть такой набор команд трансформации полным.

Для алфавита ЦА наиболее удобны цифровые в буквальном смысле системы, универсальность которых обосновывается в следующей главе. И в этой связи возникает вопрос, образуют ли арифметические действия полный набор для преобразования чисел? Решение этого вопроса не совсем очевидно. (Попробуйте указать арифметические действия над числами 7 и 5, приводящие к числу 3 или дающие из числа 1 число 6.)

А не существует ли способ, допускающий произвольную трансформацию чисел? Такое преобразование может описываться понятием функции. Значения некоторых функций могут быть получены из значений аргумента с помощью известных математических действий. Функцию удобно задавать аналитически, в виде символической записи этих действий, но это возможно не для любой функции. Более универсально табличное задание функций. Любую такую таблицу можно разместить в ЗУ ЭВМ, а для того чтобы ею пользоваться, достаточно единственной команды, позволяющей перемещать операнд из одной ячейки ЗУ (включая РОНы и регистры внешних по отношению к ЭВМ устройств) в другую. Такие команды называются *командами пересылки*. В общем случае они должны содержать два адреса: ячеек источника и приемника операнда.

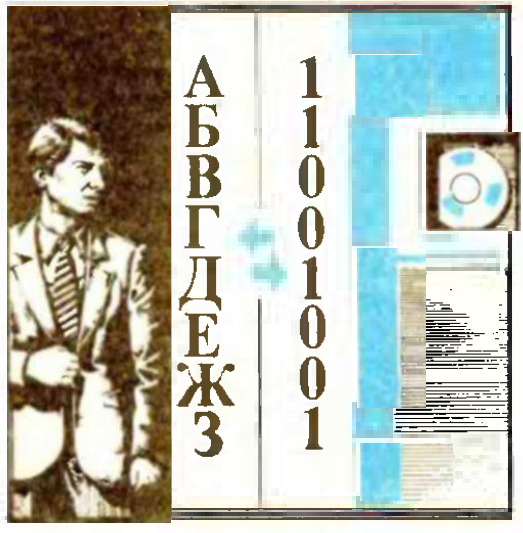
Представим себе теперь, что значения функции размещены в ячейках ОЗУ с адресами, равными соответствующим значениям аргументов. Тогда для нахождения значения функции достаточно поместить соответствующий аргумент в РОН и использовать его как косвенный адрес операнда-источника в команде пересылки. В результате ее выполнения значение функции окажется в ячейке (или РОН) по указанному в команде второму адресу. Работа с функциями, заданными таблично,— одно из важнейших применений косвенной адресации.

Несмотря на свою простоту, описанный механизм становится малоэффективным при увеличении длины операндов. Так, для

двухместных операций, даже, если операнды состоят из одной десятичной цифры, в ЗУ необходимо отвести 100 (10×10) ячеек для хранения значений функции при всех возможных комбинациях операндов. Естественное стремление к повышению быстродействия приводит к тому, что ЭВМ использует параллельное представление информации в виде слов (чисел), состоящих из нескольких символов (цифр). В этом случае затраты памяти становятся непомерно велики.

К счастью, можно поступить проще. Запись многоразрядных чисел обычно использует те же цифры, что и для одноразрядных. Поэтому произвольное преобразование таких чисел можно производить «цифра за цифрой», т. е. оно сводится к рассмотренному уже преобразованию цифр.

Оказывается, это можно сделать, не прибегая к таблицам, а в «аналитическом» виде с помощью конечного числа операций, называемых *логическими*. Устройства, в которых технически осуществлено выполнение этих операций, — *логические элементы* — являются «кирпичиками», из которых строится любой ЦА, в том числе и ЭВМ. Наряду с передачей управления и пересылками возможность выполнять логические операции в ОУ с помощью специальных команд придает автоматам, использующим принципы программного управления и хранимой в памяти программы, качество универсального ЦА, т. е. ЭВМ.



КОДИРОВАНИЕ

Распространено ошибочное мнение, что ЭВМ — быстродействующий арифмометр. В действительности же она вычислительная в том смысле, что оперирует непосредственно только с числами. А нам уже известно, что эти операции не всегда сводятся к арифметическим. Числа - удобная форма представления информации, природа которой может быть самой различной: от состояний датчиков в системах управления до сообщений на привычном нам естественном языке.

Как представить всю эту информацию с помощью чисел, каковы наиболее удобные формы представления самих чисел, наконец, как придать обработанной информации удобную для использования форму - тема этой главы. В ней будут рассмотрены *кодирование* (представление дискретной информации в стандартных символических формах) и *декодирование,* т. е. обратный процесс.

Системы счисления

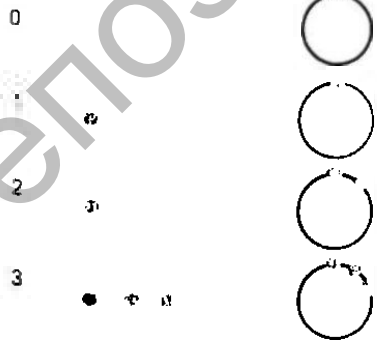
То, что любая информация может быть выражена числами, выдвигает на первый план проблему кодирования самих чисел. Примечательно, что эта проблема весьма тщательно разработана всем ходом развития нашей цивилизации. «Все есть число»,— говорили пифагорийцы, подчеркивая практическую роль чисел в человеческой деятельности. Известно множество способов представления чисел. В любом случае число изображается символом или группой символов (словом) некоторого алфавита. Будем называть такие символы *цифрами*, символические изображения чисел — *кодами*, а правила их получения — *системами счисления* (кодирования).

Простейшая и самая древняя система счисления использует для записи любых чисел всего один символ (рис. 3.1). Длина записи числа при таком кодировании прямо связана с его величиной, что роднит этот способ с геометрическим представлением чисел в виде отрезков. Сами того не сознавая, этим кодом пользуются малыши, показывая пальцами свой возраст. С ним же тесно связан *код «1 из n»* (рис. 3.2). Изображение чисел точками на оси, широко используемое в математике, фактически представляет их запись в коде «1 из *n*». Показания стрелочных часов, измерительных приборов определяются одной позицией из *n* возможных. Нажатием кнопки в лифте указывается нужный номер этажа в коде «1 из *n*».

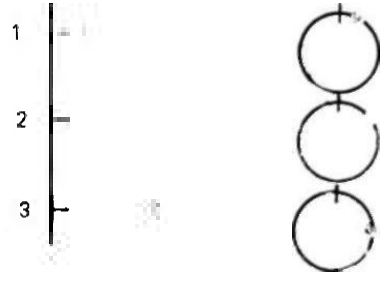
Если символ в записи чисел играет роль, зависящую от его позиции, то системы счисления, обладающие этим свойством, называются позиционными. Из всех позиционных систем наибольший практический интерес представляют те, в которых получение кода числа (кодирование) и обратный процесс (декодирование), описываются несложными правилами, например сводятся к арифметическим действиям.

Простейшие системы такого рода называются *взвешенными*. В них процедура декодирования осуществляется следующим образом: каждая цифра кода умножается на коэффициент *(вес)* позиции, и все результаты складываются (рис. 3.3). Веса отдельных позиций обычно убывают, если читать цифры кода слева направо. Левая цифра кода числа обычно читается и произносится первой, и если она самая «весомая», то уже по ней можно примерно оценить величину всего числа. Поэтому крайняя левая позиция кода числа называется старшей, а крайняя правая — младшей.

При изображении чисел в простейшем коде вместо точки может быть использован произвольный символ, например * или 1.



Если задано «начало отсчета» — место, начиная с которого пишутся символы простейшего кода, то все его символы, кроме последнего, можно опустить. Величина числа, таким образом, определяется позицией, которую занимает единственный символ по отношению к началу записи.



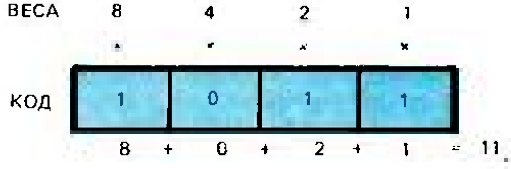


Рис. 3.3. При декодировании кода 1011 в системе с весами 8—4—2—1 получается число 11. Говоря строго, 11 — это тоже код. Обычно мы отождествляем сами числа с их записями в десятичной системе счисления (кодирования!), к которым привыкли с детства. Заметим, что и все необходимые для декодирования вычисления проведены по правилам обычной, т. е. десятичной арифметики.

Веса отдельных позиций можно выбрать совершенно произвольно, получая таким образом различные системы счисления. Не все из них одинаково удобны. С особенностями выбора весов познакомимся при рассмотрении десятичной системы счисления.

3.3. Как мы считаем?

Изобретение десятичной системы счисления приписывают древним арабам, развитие — индусам. Появление ее в Европе датируется примерно 1200 г. н. э. Десятичными цифрами выражаются время, номера домов и телефонов, цены, бюджет, показания приборов, на них базируется метрическая система мер.

В чем же секрет такого успеха десятичной системы, конкуренции с которой не выдержало подавляющее большинство остальных систем?

Процесс декодирования десятичных чисел мы обычно осуществляем «автоматически». Запись любого числа, скажем 1987, мы расшифровываем так: одна тысяча, плюс 9 сотен, плюс 8 десятков, плюс 7 единиц. Следовательно, наша система — взвешенная, а веса в ней — последовательные целые степени числа 10 ($1=10^0$, $10=10^1$, $100=10^2$, $1000=10^3$, ...). Главная особенность этого выбора весов состоит в том, что между всеми возможными числами и кодами существует взаимно однозначное соответствие, а арифметические операции над числами производятся при помощи небольшого числа несложных правил.

Не будем перечислять эти правила, они известны любому первокласснику. В их основе две таблицы — умножения и сложения для всех возможных комбинаций одноразрядных чисел (цифр) и *правило переноса:* если в результате сложения двух цифр получается число, которое больше или равно 10, то его можно записать только с помощью двух цифр с соответствующими весами.

Именно в простоте арифметических действий, операций кодиро-

вания и декодирования кроется успех десятичной системы счисления. Правила обращения с десятичными числами, изучаемые еще в детском возрасте, в результате повседневной практики усваиваются настолько крепко, что мы пользуемся ими скорее подсознательно. По этой причине многие люди даже не догадываются о существовании других систем счисления.

3.3. А почему десятичная?

Попробуем обобщить наши представления о десятичной системе счисления. Любой код, обозначаемый последовательностью произвольных цифр, например 748, интерпретируется нами как число, равное

$$7 \cdot 10^2 + 4 \cdot 10^1 + 8 \cdot 10^0.$$

Естественным обобщением такого представления чисел является кодирование дробей: запись вида 0,365 означает не что иное, как

$$\frac{365}{1000} = \frac{3}{10} + \frac{6}{100} + \frac{5}{1000}.$$

Итак, в общем случае символическая запись

$$a_n \ldots a_1 a_0,\ a_{-1} a_{-2} \ldots a_{-m}$$

расшифровывается как

$$a_n \cdot 10^n + \ldots + a_1 \cdot 10 + a_0 + a_{-1} \cdot 10^{-1} + \ldots a_{-m} \cdot 10^{-m},$$

а система счисления называется *позиционной с основанием 10*, или просто десятичной.

Очевидно, что имеют полное право на существование позиционные системы с произвольным основанием, а в некоторых отношениях, если отвлечься от привычки, они могут оказаться и удобнее десятичных.

В системе счисления с произвольным основанием b запись

$$a_n \ldots a_1 a_0,\ a_{-1} a_{-2} \ldots a_{-m}$$

соответствует десятичному числу, которое находится как

$$a_n \cdot b^n + \ldots a_1 \cdot b + a_0 + a_{-1} \cdot b^{-1} + a_{-2} \cdot b^{-2} + \ldots a_{-m} \cdot b^{-m}$$

по правилам обычной (десятичной) арифметики.

Так,

$$123_8 = 1 \cdot 8^2 + 2 \cdot 8 + 3 = 83$$
$$1010_2 = 1 \cdot 2^3 + 1 \cdot 2^1 = 10,$$

где индексы в записи чисел означают используемую систему счисления.

Обратный перевод обычно осуществляют т. н. *методом деления*, в ходе которого нужно выписать остатки от последовательного деления десятичного числа на величину основания b и прочитать их, начиная с последнего. Так, производя последовательное деление числа 19 сначала на 8, а затем на 2:

	остаток
$19 : 8 = 2$	3
$2 : 8 = 0$	2

	остаток
$19 : 2 = 9$	1
$9 : 2 = 4$	1
$4 : 2 = 2$	0
$2 : 2 = 1$	0
$1 : 2 = 0$	1

находим, что $19 = 23_8 = 10011_2$.

Для перевода десятичных дробей (чистых, т. е. без целой части) используют *метод умножения*. При этом исходная дробь умножается на величину b. Целая часть результата представляет собой первую цифру после запятой в искомой дроби. Дробная часть результата снова умножается на b и т. д. Так, например,

$$0{,}15625 \cdot 8 = 1{,}25, \qquad 0{,}75 \cdot 2 = 1{,}5,$$
$$0{,}25 \cdot 8 = 2{,}0; \qquad 0{,}5 \cdot 2 = 1{,}0$$

и, следовательно, $0{,}15625 = 0{,}12_8$, а $0{,}75 = 0{,}11_2$.

С целью перевода в двоичную систему десятичных чисел не очень большой длины можно просто подобрать различные степени числа 2, дающие в сумме исходное число. Так, $10{,}25 = 2^3 + 2^1 + 2^{-2} = 1010{,}01_2$.

Арифметические операции над числами в недесятичных системах счисления производятся по тем же правилам, что и над обычными числами. Небольшая тренировка позволяет обходиться и без таблиц сложения и умножения: все необходимые варианты легко воспроизводятся в уме. Главное при этом — помнить, что перенос в следующий разряд возникает тогда, когда результат действия над двумя цифрами равен или больше основания. Ниже приведены примеры выполнения четырех арифметических действий над числами в восьмеричной системе счисления.

364	525	364	525 | 37
+ 525	− 364	× 525	− 37 | 13
1111	141	2304	135
		+ 750	135
		2304	0
		242404	

Рекомендуем читателю внимательно проанализировать «необычные» ситуации и убедиться в правильности выполненных операций путем перевода исходных чисел и результатов в десятичную систему.

3.4. Алфавит для ЭВМ, или если бы первоклассником была машина

Двумя основными функциями ЦА являются хранение и преобразование символов (см. гл. 2). Обсудим особенности реализации этих функций в различных системах счисления. Прежде всего заметим, что в качестве *запоминающего элемента* (ЗЭ) для хране-

ния любой цифры позиционной системы счисления с основанием b может служить произвольное устройство, имеющее ровно b устойчивых состояний, каждое из которых ставится в соответствие определенной цифре. С этой целью могут использоваться обычные механические переключатели, имеющие нужное число положений. В арифмометрах роль ЗЭ играют вращающиеся шестеренки, в которых фиксируется ровно десять устойчивых положений. Недостаток всех механических ЗЭ — низкое быстродействие.

Создание электронных ЗЭ, имеющих много устойчивых состояний, затруднено. В то же время можно предложить немало способов для реализации ЗЭ с двумя устойчивыми состояниями: ток в катушке идет или не идет, конденсатор заряжен или разряжен и т. п. Правда, эти состояния не вполне устойчивы: возбужденный в катушке ток в конце концов прекращается, а конденсатор — разряжается. Следовательно, ЗЭ на их основе могут хранить информацию лишь ограниченное время. Этого недостатка лишены, например, ЗЭ из ферритовых колец. Зафиксировав такое колечко в пространстве, его можно намагнитить в направлении по часовой стрелке или наоборот. Любое такое состояние сохраняется неограниченно долго без притока энергии извне.

Итак, если состояния несложных электронных ЗЭ соответствуют двум символам, то для хранения информации целесообразно ее кодирование в двоичной системе счисления. Может показаться, что при таком выборе выигрыш в простоте ЗЭ достигается слишком дорогой ценой. Ведь запись произвольного числа в двоичной системе по сравнению с любой другой будет самой длинной, т. е. для хранения числа потребуется очень много ЗЭ. На самом деле все обстоит не так уж плохо. Различие в требуемом числе ЗЭ чисто количественное, а сложность ЗЭ в недвоичных системах счисления возрастает качественно. Даже если предположить, что сложность ЗЭ прямо пропорциональна основанию системы счисления, то затраты на оборудование для хранения двоичных чисел едва ли не минимальны. В этом смысле теоретически наиболее выгодна троичная система счисления.

С двоичной системой связаны и другие серьезные преимущества. Она обеспечивает максимальную помехоустойчивость в процессе передачи информации как между отдельными узлами автоматического устройства, так и на большие расстояния. В ней предельно просто выполняются арифметические действия, ведь таблицы сложения и умножения занимают всего лишь восемь строк:

$$\begin{array}{ll} 0+0=0 & 0\cdot 0=0 \\ 0+1=1 & 0\cdot 1=0 \\ 1+0=1 & 1\cdot 0=0 \\ 1+1=10 & 1\cdot 1=1 \end{array}$$

Нетрудно себе представить, как упростилось бы обучение арифметике малышей, если бы мы пользовались двоичной систе-

мой счисления. Наконец, в двоичной системе наиболее просто описываются и реализуются логические операции.

Благодаря таким особенностям двоичная система стала стандартом при построении ЭВМ, однако уместно заметить, что она издавна была предметом пристального внимания. Вот что писал выдающийся французский математик Пьер Симон Лаплас (1749—1807) об отношении к двоичной (бинарной) системе уже упоминавшегося великого немецкого математика Г. Ф. Лейбница: «В своей бинарной арифметике Лейбниц видел прообраз творения. Ему представлялось, что единица представляет божественное начало, а нуль — небытие, и что высшее существо создает все сущее из небытия точно таким же образом, как единица и нуль в его системе выражают все числа». Оставим в стороне свойственные тому времени религиозные заблуждения и согласимся с удивительной универсальностью алфавита, состоящего всего из двух символов.

3.5. **Бит, байт, слово**

Сопоставим ряды натуральных чисел в двоичной и десятичной системе счисления, ограничившись четырьмя разрядами для записи двоичных чисел.

десятичное число	двоичное число	десятичное число	двоичное число
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Приведенное соответствие можно построить двумя способами: переводя десятичные числа в двоичные или же последовательно прибавляя единицу к предыдущему числу столбца. Нетрудно увидеть следующую закономерность: добавление единицы к двоичному числу всегда изменяет его младшую цифру на противоположную, а каждая последующая цифра изменяется лишь в том случае, когда предыдущая изменила свое значение с 1 на 0. Именно в этом случае происходит перенос в старший разряд и именно эта особенность используется при построении *двоичных счетчиков* (см. гл. 6).

Запись двоичных чисел в примере может показаться несколько странной: мы пишем 0001, 0010, 0100 и т. п. вместо 1, 10, 100, сохраняя «незначащие» нули в старших разрядах. Этим подчеркивается параллельный характер обработки информации в ЭВМ. Человек осуществляет арифметические операции над числами последовательно, цифра за цифрой; когда при этом встречаются нули в старших разрядах, то всегда можно разобраться, «значащие» ли они, и в противном случае закончить вычисления

или их этап. ЭВМ производит операции над числами параллельно сразу во всех разрядах, поэтому во всех разрядах всегда должно быть что-то записано, даже если это «незначащий» нуль.

Уже отмечалось, что совокупность символов, обрабатываемых автоматом параллельно, называется *словом*. Слово, с которым оперирует ЭВМ, даже если это закодированная информация нечисловой природы, всегда можно интерпретировать как двоичное число. Один разряд двоичного числа называется *битом* (сокращение английского словосочетания Binary digIT – двоичная цифра). Большие ЭВМ оперируют словами в 32 бита и более, а для большинства микропроцессоров характерная длина слов — 16 бит, хотя до сих пор распространены и 8-разрядные процессоры, а совсем недавно стали появляться 32-разрядные.

Может показаться, что конечный диапазон представления чисел в ЭВМ ограничивает ее вычислительные возможности. На самом деле короткая длина слова приводит только к снижению быстродействия ЭВМ: обработка достаточно больших чисел ведется последовательно-параллельным способом. При этом сами числа представляются несколькими машинными словами, а для выполнения операций над ними необходимо составить специальные программы.

Для ЭВМ и микропроцессоров характерна возможность производить действия и над укороченными операндами. Это во многом обусловлено тем, что для кодирования каждого символа текстовой информации чаще всего требуется совокупность из 8 бит, называемая *байтом*. Эффективность обработки текстов повышается, если в машинном слове укладывается целое число байтов и обеспечена возможность отдельной работы с каждым из них.

3.6. Особенности машинной арифметики

Представление чисел словами конечной длины накладывает особый отпечаток на характер арифметических действий. В качестве примера рассмотрим выполнение сложения в ЭВМ, оперирующей байтами (рис. 3.4). Возникающая в этом примере единица переноса из старшего разряда не «вписывается» в слово, содержащее результат. Такая ситуация называется *переполнением разрядной сетки* машины. Она может привести к получению неверного результата, поэтому УУ строится таким образом, что в случае переполнения всегда происходит изменение его состояния. Внутреннее состояние обычно отражается т. н. *словом состояния процессора* (ССП). ССП содержит информацию об особых результатах выполнения каждой команды и хранится в специальном регистре УУ. Отдельный его бит, обозначаемый буквой *C* (от англ. *Carry* — перенос), устанавливается в единичное состояние как раз при наличии переноса из старшего разряда.

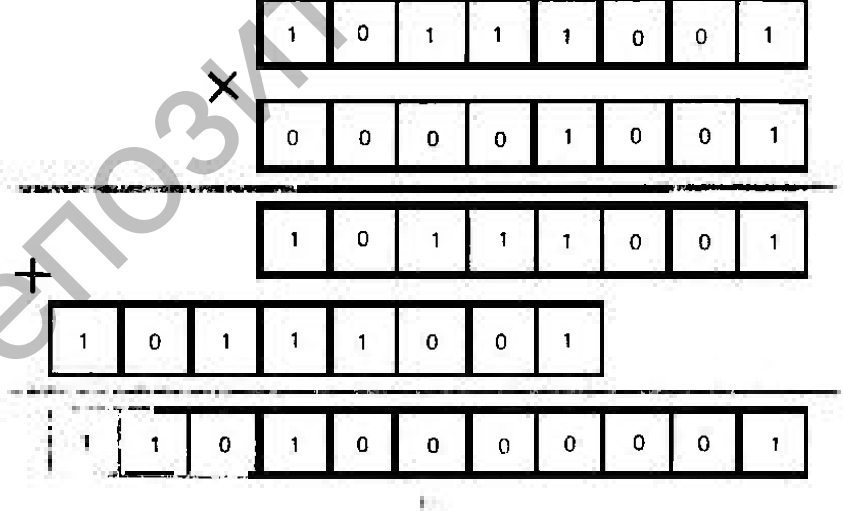
Обычно при переполнении управление передается участку программы, который использует для представления результата

При сложении чисел 153_{10} и 106_{10}, представленных байтами, перенос, возникающий в третьем разряде, «путешествует» влево и оказывается за пределами байта результата. Аналогичная ситуация может возникнуть и при любой другой (конечной) длине слова. Если не принять специальных мер, единица переноса из старшего разряда будет утеряна, что приведет к ошибочному результату.

более, чем одно слово, или сигнализирует о возникшей ситуации человеку.

При умножении (рис. 3.5) результат может занимать несколько дополнительных разрядов. Многие ЭВМ вообще не имеют команды умножения. Оно осуществляется специальными стандартными программами, содержащими команды сложения и сдвига. При

В ходе умножения двоичных представлений чисел 185_{10} и 9_{10} сразу три разряда результата не укладываются в однобайтовом слове.



выполнении каждой из таких команд переполнение может анализироваться по С-биту.

Умножение производится значительно эффективнее, если ЭВМ обладает *командами расширенной арифметики* (РА). Одна из команд РА выполняет операцию умножения двух операндов длиной в машинное слово с размещением результата в двух машинных словах. В этом случае говорят, что результат имеет *двойную точность*. Двух слов всегда хватает для представления произведения однословных операндов, поэтому после выполнения такой команды бит переполнения принимает нулевое состояние.

Команда РА для выполнения деления использует два слова для представления делимого и по одному — для делителя, частного и остатка.

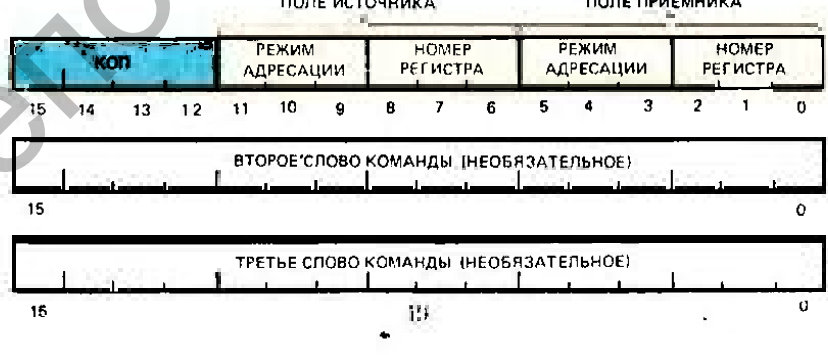
РА и особые команды для работы над т. н. *числами с плавающей запятой* значительно повышают вычислительные возможности ЭВМ.

3.7. Как кодируются команды?

Считывая очередную команду программы из памяти, процессор ЭВМ должен получить исчерпывающие сведения о том, как эту команду выполнять. В их числе — тип операции, число операндов с подробными указаниями на способ их адресации, число слов памяти, занимаемых текущей командой.

Кодирование команд рассмотрим на конкретном примере шестнадцатиразрядных ЭВМ типа СМ-4, ДВК и им подобных. Процессоры этого семейства допускают явную адресацию в одной команде двух операндов. Типичные двухадресные команды — сложение, вычитание, пересылка, сравнение. Каждая такая команда в общем случае занимает три последовательных слова ОЗУ (рис. 3.6).

Рис. 3.6. Первое слово двухадресной команды содержит код операции (КОП) и номера РОН с указаниями на используемый способ адресации для каждого из операндов. Один из них называется операндом-источником. Двоичный номер соответствующего ему РОН помещается в битах 6...8. Вместе с битами 9...11, определяющими один из восьми возможных способов адресации, они образуют т. н. поле источника. Биты 0...5, составляющие поле приемника, выполняют аналогичные функции для второго операнда (приемника).



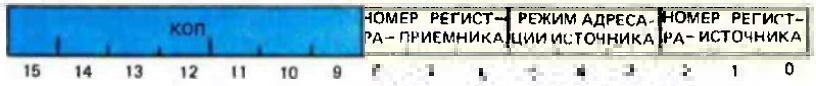


Рис. 3.7. Регистровые команды имеют семибитный КОП. В их числе упоминавшиеся в предыдущем параграфе команды РА, для которых биты 0...5 указывают на РОН и характер адресации источника, а биты 6...8 определяют регистр-приемник. В случае команды деления (КОП-0111001) в регистр-приемник (его номер обязан быть четным) и следующий за ним по счету РОН помещается 32-разрядное делимое, а после выполнения операции — частное и остаток. Для некоторых других команд этой группы биты 0...5 исполняют особые функции.

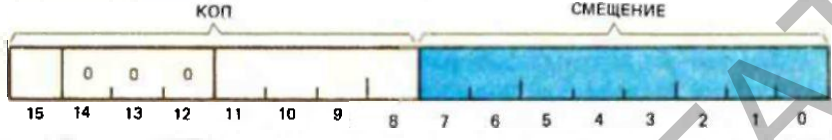


Рис. 3.8. КОП команд перехода по условию занимает весь старший байт. Биты 8...11 кодируют условие перехода в зависимости от значения отдельных битов ССП. При выполнении заданного условия осуществляется передача управления к команде, отстоящей от текущей на число слов, определяемое смещением.

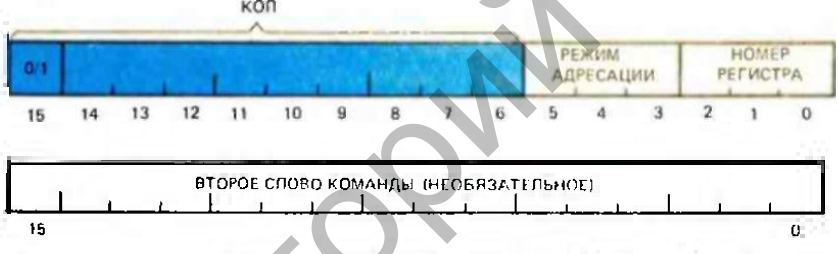


Рис. 3.9. Одноадресные команды обладают 10-битным КОП. Поля «режим» и «номер регистра» определяют характер адресации (аналогично двухоперандным командам).

Названия «источник» и «приемник» для операндов соответствуют их роли при выполнении команды пересылки, которая «копирует» *операнд-источник* в регистр или ячейку ОЗУ, занимаемую *операндом-приемником*. Операнд-источник сохраняется неизменным и при выполнении большинства других команд, результат которых располагается на месте операнда-приемника.

Необязательные второе и третье слова двухадресных команд могут нести дополнительную информацию об операндах при некоторых способах адресации.

Команды кодируются сложнее, чем числа. Отдельные биты чисел используются ЭВМ совершенно одинаково. Напротив, группы битов *(поля)* и даже отдельные биты в записи команд могут играть совершенно различную роль. Так, бит 15 в КОП для боль-

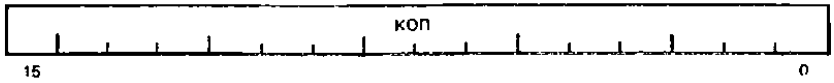


Рис. 3.10. Команды управления состоят только из КОП. Одна из таких команд с КОП=000...00 приводит к прекращению выполнения программы, т. е. к остановке процессора

шинства команд играет роль указателя длины операндов: если он равен нулю, команда оперирует со словами, в противном случае — с байтами. Оставшиеся три бита интерпретируются как двоичный номер, указывающий на конкретную операцию. Поскольку три бита дают $2^3 = 8$ различных двоичных чисел, то имеется возможность кодирования шестнадцати различных команд (по восемь), оперирующих с байтами и словами.

Может сложиться впечатление, что рассматриваемые процессоры рассчитаны на выполнение только шестнадцати команд. В действительности их число во много раз больше благодаря механизму т. н. *кодов расширения*. Дело в том, что необходимое на практике число двухадресных команд меньше шестнадцати. Поэтому некоторые из шестнадцати возможных комбинаций четырех старших битов — коды расширения — указывают процессору на то, что соответствующие команды не двухадресные и имеют совершенно иной формат.

В числе таких кодов — 0000, 1000, 0111, 1111. Некоторые из этих кодов приводят к группе т. н. регистровых команд (рис. 3.7.). Аналогично получается группа команд условных переходов, формат которых приведен на рисунке 3.8.

Некоторые из кодов расширения команд перехода приводят к группе одноадресных команд, формат которых представлен на рисунке 3.9. В нее входит, например, *команда очистки* содержимого адресуемой ячейки памяти, т. е. установки всех ее битов в нулевое состояние.

Наконец, коды расширения однооперандных команд приводят к группе команд, использующих для записи КОП все биты командного слова (рис. 3.10). Такие команды не имеют операндов, они используются для управления работой процессора, в частности для изменения состояния ССП.

3.8. Зачем нужны иные системы кодирования?

Десятичная система счисления малоприемлема для ЭВМ, а двоичное кодирование неудобно для человека, занимающегося созданием программ. Усилия специалистов по вычислительной технике привели к тому, что большинство программ для современных ЭВМ пишутся не в двоичной и даже не в десятичной системе счисления, а на *языках высокого уровня*, значительно более удобных для человека. Ведутся работы в направлении обеспечения диалога с ЭВМ на естественном, человеческом языке, т. е. работы по автоматическому переводу информации, курсирующей между

человеком и ЭВМ в процессе их диалога. Значительную роль в обеспечении такого перевода играют сами ЭВМ как универсальные устройства обработки информации.

Тем не менее знание особенностей двоичного представления чисел и команд необходимо не только разработчикам и обслуживающему персоналу ЭВМ, но и программистам. Понимание тонкостей *машинного языка* позволяет получить более эффективные программы, даже если они пишутся на языке высокого уровня.

А нельзя ли найти удобную для человека форму восприятия двоичной информации? На первый взгляд кажется, что для этой цели достаточно снабдить ЭВМ устройством для преобразования двоичных чисел в десятичные, и наоборот. Однако при кодировании команд отдельные биты могут играть определенную, индивидуальную роль. В процессе перевода кода команды в десятичное число значения этих битов становятся трудноосязаемыми, поэтому запись десятичного кода нужной команды сразу, минуя ее двоичное представление, становится весьма трудоемкой. Напротив, чтобы выяснить смысл команды, представленной десятичным числом, придется обязательно перевести его в двоичную систему счисления.

В определенной мере указанных недостатков лишены *двоично-десятичные системы кодирования*. В них каждой десятичной цифре ставится в соответствие двоичный код, состоящий чаще всего из четырех символов. В двоично-десятичном коде десятичного числа вместо каждой десятичной цифры последовательно записываются соответствующие им четверки битов. Для обратного перевода двоичная запись разбивается на группы по четыре бита, и каждая группа заменяется соответствующей десятичной цифрой. Отдельный бит двоично-десятичного кода влияет при этом только на единственный знак десятичного эквивалента.

Однако двоично-десятичному кодированию свойственна избыточность. Оно требует больше места в памяти ЭВМ, поэтому применение двоично-десятичных кодов ограничивается устройствами ввода-вывода данных.

3.9. Эсперанто для человека и ЭВМ

Если произвольное двоичное число разбить на группы по n битов, то каждой такой группе можно поставить в соответствие цифру системы счисления с основанием $b = 2^n$. В частности, любой тройке битов будет соответствовать единственная *восьмеричная* ($8 = 2^3$) цифра, и, наоборот, произвольной восьмеричной цифре можно поставить в соответствие единственную тройку битов (триаду). Этим определяется простая взаимосвязь между записями чисел в двоичной и восьмеричной системах счисления.

С целью перевода в восьмеричную систему двоичное число разбивается на триады (справа налево). Если при этом в крайней слева группе окажется менее трех двоичных цифр, недостающие позиции заполняются нулями. Ниже приведены таблица соответ-

ствия двоичных триад восьмеричным цифрам и примеры перевода из двоичной системы в восьмеричную, и наоборот.

Двоичная триада	Восьмеричный эквивалент
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

$110\,010\,111\,101_2 = 6275_8$;
$1304_8 = 001\,011\,000\,100_2$;
$1\,011\,010\,011\,110\,100_2 = 132364_8$;
$0\,111\,001\,101\,100\,111_2 = 071547_8$.

Два последних примера характерны для ЭВМ, оперирующих шестнадцатиразрядными словами. Восьмеричные записи таких слов всегда содержат шесть цифр, старшая из которых может принимать лишь значения 0 и 1.

Совершенно аналогично, на основе таблицы соответствия четверок битов (тетрад) и *шестнадцатеричных* цифр, устанавливается взаимосвязь между любыми двоичными словами и их шестнадцатеричными эквивалентами:

Двоичная тетрада	Шестнадцатеричный эквивалент
. . . .	...
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	*A*
1 0 1 1	*B*
1 1 0 0	*C*
1 1 0 1	*D*
1 1 1 0	*E*
1 1 1 1	*F*

$1101\,1110\,0101\,1010_2 = \mathrm{DE5A}_{16}$;
$1101111100_2 = 37\mathrm{C}_{16}$;
$3\mathrm{A}_{16} = 0011\,1010_2 = 111010_2$.

Запись двоичных чисел в любой из этих форм компактнее, проще воспринимается, а значит, достаточно удобна для человека. Взаимосвязь отдельных битов двоичного числа и символов используемой формы локализована: значение отдельного бита влияет на единственный символ, и наоборот, каждый символ связан только с небольшой группой из трех или четырех битов. Наконец, алгоритмы перевода двоичного числа в одну из рассмотренных форм (и наоборот) очень просты и для человека, и для ЭВМ.

Восьмеричная или шестнадцатеричная системы счисления могут выступать в качестве простейшего языка общения человека с ЭВМ, достаточно близкого как к привычной для человека десятичной системе счисления, так и к двоичному «языку» машины. Запись чисел в системах счисления с основанием $b = 2^n$ можно рассматри-

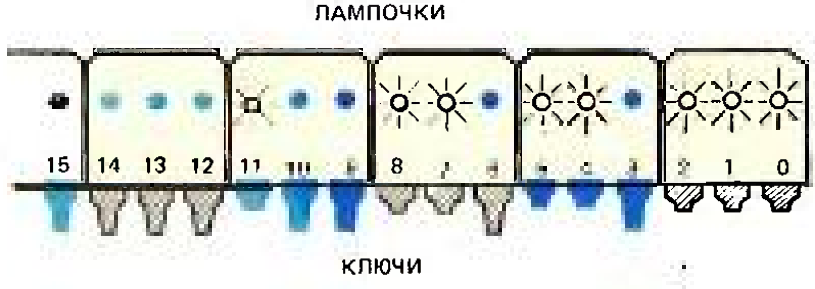


Рис. 3.11 Пульт шестнадцатиразрядной ЭВМ использует двоичную систему: 0 — ключ (тумблер) опущен, лампочка не горит, 1 — ключ поднят, лампочка горит. Группировка этих элементов в триады (ключи обычно имеют чередующуюся от группы к группе окраску) существенно упрощает как восприятие двоичной информации в восьмеричной форме, так и ввод восьмеричных чисел в двоичном виде.

вать как своеобразные формы представления двоичной информации, а при $n = 3, 4$ они близки к соответствующим десятичным числам с точки зрения компактности.

По этой причине использование восьмеричных и шестнадцатеричных кодов машинных команд, адресов ОЗУ, операндов широко распространено как на этапах составления несложных программ для микро-ЭВМ, их отладки, ручного ввода-вывода данных, так и на этапах разработки, создания, настройки и обслуживания вычислительных систем. Не случайно пульт управления большинства ЭВМ предполагает ввод, вывод (или индикацию) машинных слов именно в одной из таких систем счисления, чаще в восьмеричной (рис. 3.11).

На этом рисунке не показаны вспомогательные органы, которые индицируют состояние ЭВМ и ее узлов, инициируют ввод в машину адреса ячейки ОЗУ или ее содержимого и т. п.

3.10. Представление отрицательных чисел дополнительными кодами

Одной из причин появления в математике отрицательных чисел была потребность сделать всегда выполнимой операцию вычитания. Выражение $2-5=$, бессмысленное на множестве натуральных чисел, принимает вполне определенное значение, равное -3, на множестве целых чисел. Сама запись отрицательных, как и любых других чисел, по существу представляет собой некоторый код, расшифровываемый по особым правилам. Так,— 2 есть число, равное 2 по величине (модулю), но противоположное ему по знаку.

Операции над отрицательными числами определены так, чтобы они подчинялись тем же законам, что и натуральные числа. Как следствие из сказанного, для любого числа $y+(-y)=0$, а опе-

рацию вычитания двух чисел всегда можно свести к сложению согласно известной формуле $x - y = x + (-y)$. К сожалению, в конкретных вычислениях не обойтись без операции вычитания: несмотря на то что $2 - 5 = 2 + (-5)$, мы находим разность $5 - 2 = 3$ и присваиваем ей знак минус, потому что в исходном выражении из меньшего (по модулю) числа вычиталось большее.

Однако свести операцию вычитания к сложению все-таки можно благодаря явлению переполнения. Это реализуется при помощи специального способа кодирования отрицательных чисел — представления их в так называемом *дополнительном коде*.

Сущность способа нетрудно понять, изучая работу *реверсивного счетчика*. Подходящий счетчик имеется, например, в большинстве магнитофонов и служит в качестве индикатора расхода ленты. Его показания увеличиваются либо уменьшаются в зависимости от направления движения ленты. Интересующая нас особенность состоит в том, что при возрастании показаний счетчика до максимального, например до 999, следующими его состояниями будут 000, 001 и т. д. Происходит то, что мы называем переполнением разрядной сетки: $999 + 1 = 1000$, но для изображения единицы переноса в нашем счетчике не хватает одного разряда. При обратном движении ленты показания счетчика убывают, а после состояния 000 следует 999, 998, ... Отвлекаясь от конкретного устройства счетчика, логику изменения его состояний можно представить, как показано на рис. 3.12.

Но ведь последовательное вычитание единицы из некоторого числа, после достижения нуля, должно давать отрицательные числа: -1, -2, ... Нельзя ли связать каким-либо образом состояние 999 с числом -1? Может быть следует рассматривать 999 как искомый код числа -1?

Проверим нашу гипотезу: $001 + 999 = 1000$, а с учетом того, что единица переполнения теряется, получаем 000. Проверяя таким же образом соотношение $y + (-y) = 0$ для других y, можно убедиться, что 998, 997, ... так же подходят в качестве кодов отрицательных чисел -2, -3, ...

Попробуем теперь сложить положительное число, допустим 5 с кодом отрицательного числа -3, т. е. с 997. Получим $5 + 997 = 1002$, а с учетом потери единицы переполнения — число 2, т. е. правильный результат, который в обычной арифметике получается вычитанием: $5 - 3 = 2$.

Для того чтобы сформулировать точное определение дополнительного кода, осталось выяснить, какие коды в кольце, представленном на рисунке 3.12, соответствуют положительным, а какие отрицательным числам, иначе возникает неоднозначность в их интерпретации. Естественно следующее решение: считать половину состояний (от 0 до 499) — кодами нуля и положительных чисел, а оставшуюся половину (500...999) — кодами отрицательных чисел.

Таким образом, дополнительный код положительного числа совпадает с этим числом, а для отрицательного числа он равен дополнению его величины до числа $x_{\text{пер}}$, возникающего при пере-

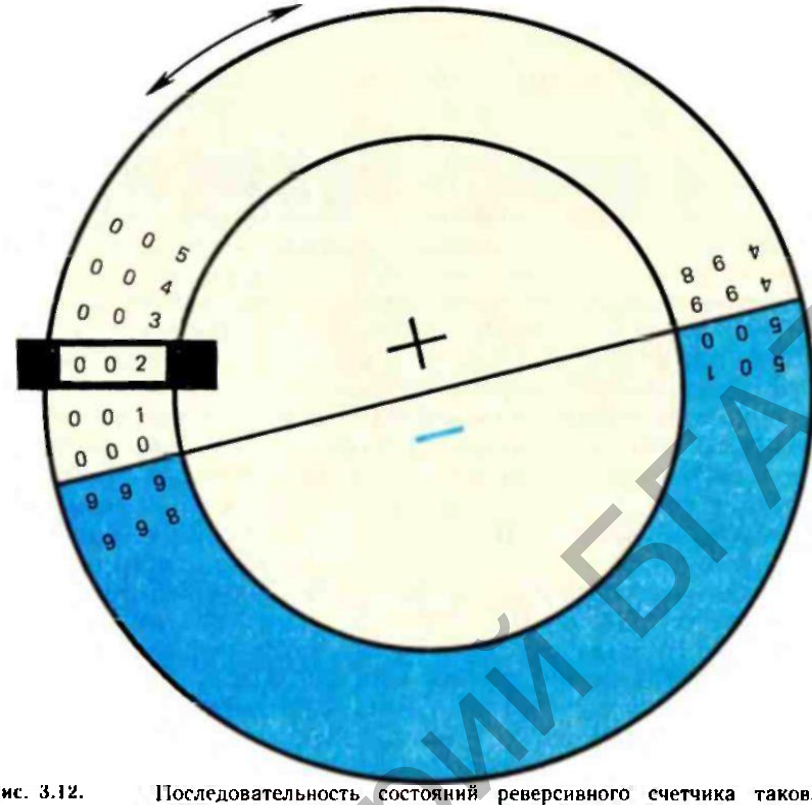


Рис. 3.12. Последовательность состояний реверсивного счетчика такова, что ее можно представить в виде замкнутого кольца из чисел. Если начальные показания счетчика 002, то в режиме вычитания следующими состояниями будут 001, 000, а затем 999, 998, ..., которые можно интерпретировать как коды отрицательных чисел -1, -2, ...

полнении используемой разрядной сетки. Очевидно, $x_{\text{пер}} = b^n$, где b — основание системы счисления, n — число разрядов. Дополнения приходится считать и при декодировании отрицательных чисел, коды которых лежат в пределах от $\frac{x_{\text{пер}}}{2}$ до $x_{\text{пер}} - 1$.

Находить дополнение числа x до $x_{\text{пер}}$, т. е. разность $b^n - x$, можно и другим способом. Обозначим через $\bar{x}$ число, полученное заменой каждой цифры числа x на ее дополнение до цифры $b - 1$. Очевидно, $\bar{x} + x = b^n - 1$. Так, $123_{10} + \overline{123}_{10} = 123 + 876 = 999$. Следовательно, $b^n - x = \bar{x} + 1$.

Такой способ особенно удобен в двоичной системе счисления, т. к. число $\bar{x}$, называемое в этом случае *инверсией x*, получается в результате простой замены в числе x единиц нулями и наоборот. Следовательно, дополнения двоичных чисел можно находить, минуя операцию вычитания.

На практике важна еще одна особенность дополнительных ко-

дов двоичных чисел. Заметим, что число $\frac{x_{\text{пер}}}{2}$ состоит из единицы в старшем бите и нулей в последующих, поэтому все дополнительные коды отрицательных чисел имеют в старшем бите единицу, а положительных чисел — нуль. Говоря другими словами, старший бит дополнительного кода является признаком знака закодированного числа *(знаковым битом)*.

Итак, представление целых чисел дополнительным кодом позволяет обойтись без операции вычитания. Действительно, для любых чисел $x - y = x + (-y)$ и при условии, что x и $-y$ представлены дополнительным кодом, их сложение приведет к правильному результату. Сам переход к дополнительным кодам также не нуждается в операции вычитания, а может быть сведен к более простым операциям инверсии и добавления единицы *(инкрементации)*. По этой причине набор команд ЭВМ, состоящий из сложения, побитной инверсии и инкрементации, достаточен для выполнения всех арифметических действий, ведь умножение и деление могут быть сведены к сложению и вычитанию, что вытекает из известных всем алгоритмов выполнения этих действий.

Диапазон представления чисел n-разрядными двоичными кодами изменяется от $(0...2^n - 1)$ для кодирования неотрицательных чисел до $(-2^{n-1}...2^{n-1} - 1)$ для представления целых чисел дополнительным кодом.

3.11. Особенности арифметики в дополнительном коде

Не следует забывать, что явление переполнения разрядной сетки машины, открывая возможность удобного представления чисел в дополнительном коде, все же является и основным источником неприятностей в машинной арифметике. Ограниченная длина слова ЭВМ приводит к необходимости следить за тем, чтобы в разрядной сетке машины укладывались не только операнды, но и результат выполняемого действия. Как это делается в случае представления положительных чисел натуральным двоичным кодом, достаточно подробно разбиралось в 3.6. Нетрудно предположить, что с переходом к представлению чисел дополнительным кодом трудности могут только возрасти, хотя бы из-за уменьшения максимальной величины допустимых чисел. Для шестнадцатиразрядных ЭВМ диапазон представления чисел в натуральном коде — от 0 до 65535, а в дополнительном — от -32768 до $+32767$.

Для того чтобы выяснить, в каких случаях операции с числами в дополнительном коде могут привести к неверным результатам, рассмотрим несколько простых примеров:

$+\begin{array}{r}0011\\0010\end{array}$	$+\begin{array}{r}3\\2\end{array}$	$+\begin{array}{r}0110\\1111\end{array}$	$+\begin{array}{r}6\\-1\end{array}$	$+\begin{array}{r}1010\\1111\end{array}$	$+\begin{array}{r}-6\\-1\end{array}$		
$\begin{array}{r}0101\\\hline\end{array}$	$\begin{array}{r}5\\\hline\end{array}$	$\begin{array}{r}10101\\\hline\end{array}$	$\begin{array}{r}5\\\hline\end{array}$	$\begin{array}{r}11001\\\hline\end{array}$	$\begin{array}{r}-7\\\hline\end{array}$		
Нет переносов в знаковый бит и из знакового бита. Результат верный.		Есть переносы в знаковый бит и из знакового бита. Результат верный.		Есть переносы в знаковый бит и из знакового бита. Результат верный.			

$\begin{array}{r} 0100 \\ +\,0110 \\ \hline 1010 \end{array}$	$\begin{array}{r} 4 \\ +\,6 \\ \hline -6 \end{array}$	$\begin{array}{r} 1010 \\ +\,0100 \\ \hline 1110 \end{array}$	$\begin{array}{r} -6 \\ +\,4 \\ \hline -2 \end{array}$	$\begin{array}{r} 1100 \\ +\,1010 \\ \hline 10110 \end{array}$	$\begin{array}{r} -4 \\ +\,-6 \\ \hline 6 \end{array}$
Есть перенос в знаковый бит. Результат ошибочный. Правильный результат должен быть 10_{10}, но для его представления в дополнительном коде не хватает разрядов.		Переносов нет. Результат верный.		Есть перенос из знакового бита. Результат ошибочный (должно быть -10_{10}).	

Анализ этих примеров позволяет сделать предположение, что признаком ошибочных ситуаций может служить наличие или отсутствие переносов в знаковый бит и из знакового бита в процессе сложения: результат ошибочен, если из двух этих переносов произошел только один. В остальных случаях, т. е. когда переносы вообще отсутствуют, или произошли сразу оба, сложение даст верный результат, т. е. представление чисел дополнительным кодом замкнуто относительно операции сложения. (Строгое доказательство этого утверждения может быть построено с учетом того, что запись двоичного числа в дополнительном коде можно интерпретировать как особый взвешенный код. Весовые коэффициенты этого кода для всех битов, кроме знакового, такие же, как у натурального двоичного кода, а знаковый бит имеет отрицательный вес, равный -2^{n-1}, где n — длина слова.)

Неблагоприятные ситуации, возникающие при вычислениях с числами в дополнительном коде, должны учитываться программистами. Обычно для этого в состав ССП ЭВМ вводят т. н. V-бит (от англ. *oVerflow* — переполнение), который автоматически устанавливается в единичное состояние, когда при выполнении текущей команды происходит перенос в старший (знаковый) бит или из него. Состояние этого бита может учитываться при выполнении команд условного перехода. Используя эти команды, можно избежать ошибок таким же образом, как это было рассмотрено в 3.6.

Отметим, что в состав ССП обычно вводят и другие биты, упрощающие работу с числами в дополнительном коде. Так, биты N и Z (от англ. *Negative* и *Zero*) устанавливаются, если результат предыдущей операции соответственно отрицательный или нулевой. Состояния этих битов также могут анализироваться командами перехода.

3.12. Некоторые другие представления отрицательных чисел

Представление отрицательных чисел в дополнительном коде наиболее распространено. Тем не менее, существуют ситуации, когда предпочтительнее использование других форм. В первую очередь это относится к представлению дробей, о чем пойдет речь в следующем параграфе. Здесь же мы кратко обсудим еще две распространенные формы записи отрицательных чисел.

1. Представление прямым кодом (в форме «знак-величина»). Оно соответствует привычной человеку форме записи чисел. Как и для дополнительного кода, первый бит n-разрядного слова знаковый. Остальные $n-1$ битов представляют величину (модуль) числа в двоичной системе счисления. Для выполнения арифметических операций с такими кодами нужны специальные программы, реализующие хорошо известные каждому человеку алгоритмы. Так, для операции сложения нужно проверить знаки обоих операндов. Если они одинаковы, то вычисляется сумма операндов и ей присваивается тот же знак. Если знаки противоположны, то из большего по модулю числа нужно вычесть меньшее и результату присвоить знак уменьшаемого. Тем не менее соответствующие этим алгоритмам программы не так уж просты. Знаковый бит должен анализироваться ими отдельно, а сами арифметические действия производиться только над остальными битами чисел. Это требует применения специальных команд. В сложении дополнительных кодов знаковый бит фигурирует наравне со всеми остальными битами, что приводит к более эффективной работе ЭВМ.

В системе прямых кодов есть интересная особенность, заключающаяся в существовании двух различных представлений нуля: 00...0 и 10...0, называемых *положительным и отрицательным нулем*. Нетрудно убедиться, что оба приведенных представления равноправны. Однако эта особенность является источником ошибок при составлении программ. Так, если приходится проверять результат каких-то вычислений на равенство нулю, то сравнение должно проводиться как с положительным, так и с отрицательным нулем, о чем легко забыть.

Важное достоинство представления целых чисел в форме «знак-величина» — простота операций кодирования и декодирования. Диапазон чисел, представленных прямым кодом, лежит в пределах от $-2^{n-1}+1$ до $2^{n-1}-1$. Для шестнадцатиразрядных ЭВМ эти границы равны -32767, $+32767$.

2. Представление *смещенным кодом* (с избытком 2^{n-1}). Диапазон представления чисел в этой системе $-2^{n-1} \leqslant x \leqslant 2^{n-1}-1$, т. е. такой же, как и в системе дополнительных кодов. Алгоритм кодирования основан на том, что число $x+2^{n-1}$, лежащее в пределах $0 \leqslant x+2^{n-1} \leqslant 2^n-1$, может быть представлено натуральным n-разрядным двоичным кодом. Число 2^{n-1} называется смещением. Для декодирования можно вычесть смещение из кода по правилам обычной арифметики. Интересно, что такая система идентична системе дополнительных кодов с точностью до инверсии старшего бита: в системе со смещением 0 означает отрицательное число, а 1 — положительное. На основании этой связи легко получить правила определения переполнения.

В таблице 3.1 приведена интерпретация четырехразрядных двоичных кодов в различных системах представления целых чисел.

Таблица 3.1

Двоичный код	Числа в различных представлениях			
	Натуральные числа	Дополнительное кодирование	Прямое кодирование	Кодирование со смещением
0 0 0 0	0	0	+0	−8
0 0 0 1	1	1	1	−7
0 0 1 0	2	2	2	−6
0 0 1 1	3	3	3	−5
0 1 0 0	4	4	4	−4
0 1 0 1	5	5	5	−3
0 1 1 0	6	6	6	−2
0 1 1 1	7	7	7	−1
1 0 0 0	8	−8	−0	0
1 0 0 1	9	−7	−1	1
1 0 1 0	10	−6	−2	2
1 0 1 1	11	−5	−3	3
1 1 0 0	12	−4	−4	4
1 1 0 1	13	−3	−5	5
1 1 1 0	14	−2	−6	6
1 1 1 1	15	−1	−7	7

3.13. Представление дробей. Числа с фиксированной и плавающей запятой

Нетрудно убедиться, что без каких-либо изменений в алгоритмах выполнения арифметических операций рассмотренные методы кодирования целых чисел пригодны для обозначения дробей. Так, если договориться, что в четырехразрядных дополнительных кодах двоичная запятая, отделяющая целую часть от дробной, расположена посередине, то целочисленное действие $5 + (-1) = 4$ можно интерпретировать как $1{,}25 + (-0{,}25) = 1{,}0$:

$$\begin{array}{r} 0101 \\ +\,1111 \\ \hline 10100 \end{array} \qquad \begin{array}{r} 5 \\ +\,-1 \\ \hline 4 \end{array} \qquad \begin{array}{r} 01{,}01 \\ +\,11{,}11 \\ \hline 101{,}00 \end{array} \qquad \begin{array}{r} 1{,}25 \\ +\,-0{,}25 \\ \hline 1{,}00 \end{array}$$

Таким образом, для представления дробей программист может мысленно располагать двоичную запятую в любой нужной ему позиции. Естественно, при выполнении сложения и вычитания двоичные запятые должны быть расположены одинаково в обоих операндах, таково же будет и положение запятой в результате. Представление дробей, при котором положение двоичной запятой задается неявно в определенном месте машинного слова, называется *представлением с фиксированной запятой*.

Неудобство такого представления проявляется при решении задач с величинами, которые могут сильно изменяться в сторону как очень малых, так и очень больших чисел. В конкретных физических, математических и других задачах диапазон изменения величин может составлять, например, от 10^{-30} до 10^{+30}. Можно убедиться, что в представлении с фиксированной запятой подошли

бы двоичные операнды длиной около 256 бит (32 байта), по 128 бит на целую и дробную части.

Однако работа ЭВМ с операндами такой длины была бы крайне неэффективной. На практике значительная часть из этих 256 бит оказалась бы равной нулю. Для очень маленьких чисел это очевидно. При работе же с очень большими числами их младшие разряды, а тем более дробная часть, обычно игнорируются. Ведь высокая точность результата любых вычислений достигается лишь тогда, когда все участвующие в них числа одинаково точны. Задание же всех величин с точностью до 256 бит (или примерно 76 десятичных цифр) — дело не только нереальное, но и бессмысленное, хотя бы потому, что многие из этих величин получаются в результате измерений не очень точными приборами. Не случайно в практических расчетах редко используют более трех значащих цифр, соответствующим образом округляя промежуточные результаты.

Наконец, устройство ЭВМ с 256-разрядным словом оказалось бы очень сложным, а в случае последовательно-параллельной обработки, например 16-разрядными словами, резко возросло бы время вычислений.

Выход из затруднения состоит в отказе от фиксированного расположения запятой. Действительно, 256-разрядные двоичные числа с запятой посередине можно записывать 128-разрядным словом, в котором двоичной запятой разрешено находиться в любой позиции («плавать»). Если двоичная запятая находится после крайнего справа бита, то диапазон представления положительных чисел будет от 0 до $2^{128}-1$. Если же она расположена перед старшим разрядом, то возможно изображение правильных положительных дробей от 2^{-128} до $1-2^{-128}$. Промежуточные положения запятой дают смешанные дроби, состоящие из целой и дробной части. Заметим, что двойная экономия битов в слове достигается ценой некоторой потери в точности представления именно таких дробей, ведь в их прежней записи значащими могли быть все 256 бит.

Реальный выигрыш в числе битов хотя и велик, но все же меньше, чем могло бы показаться. Ведь если разрешить запятой «плавать» во всех числах, то для выполнения операций над ними в записи каждого числа должна быть информация о месте расположения запятой. В тех случаях, когда эта информация выражена в записи чисел явным образом, говорят о *представлении чисел с плавающей запятой*.

Наиболее удобным образом информация о положении запятой задается, если числа представлены в т. н. *экспоненциальной форме записи*. Это значит, что число записывается в виде произведения дроби со знаком *(мантиссы)* и основания системы счисления, возведенного в степень с некоторым показателем *(порядком)*, например $3{,}143\cdot 10^{-7}$, или $-10{,}11_2\cdot 2^{101_2}$. Для удобства мантиссу обычно записывают в виде правильной дроби, у которой первая же цифра после запятой — значащая. Так, $3{,}143\cdot 10^{-7}=$

$= 0{,}3143 \cdot 10^{-6}$, а $-10{,}11_2 \cdot 2^{101_2} = -0{,}1011_2 \cdot 2^{111_2}$. Такая форма экспоненциальной записи называется *нормализованной*. В результате изображение двоичного числа с плавающей запятой состоит из следующих компонентов: знаковый бит мантиссы, мантисса, знак порядка, порядок. Положение двоичной запятой в мантиссе неявно фиксируется слева от ее первой цифры.

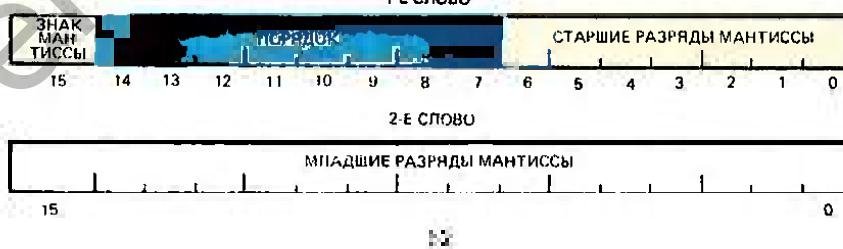
Здесь у читателя может возникнуть вопрос, какой же смысл говорить о плавающей запятой, если в мантиссе она фиксирована? Дело в том, что нормализованная мантисса фактически определяет только значащие цифры числа. Истинное положение запятой в числе после его перевода в обычную, неэкспоненциальную форму явно задает порядок. И хотя при этом для данной мантиссы и данного порядка положение запятой получается вполне определенным, то для той же мантиссы, но с другим порядком, запятая займет иное положение.

3.14. Как представляются числа с плавающей запятой в ЭВМ?

В реальных ЭВМ обычно хранятся два-три десятка старших битов, поэтому для хранения всего числа (вместе с порядком) вполне хватает одного-двух машинных слов. Так, формат чисел с плавающей запятой для ЭВМ типа ДВК и аналогичных имеет вид, показанный на рисунке 3.13, причем мантисса представляется в прямом коде, а порядок — в смещенном. Причины такого выбора комментируются в следующем параграфе.

Очевидно, что в нормализованном представлении двоичных чисел первая цифра мантиссы всегда равна 1, поэтому нет необходимости ее хранить. Учитывая этот «скрытый» бит, фактическая длина мантиссы составляет не 23, как следует из рисунка 3.13, а 24 бита. Как исключение, скрытый бит полагается равным нулю, если исходное число равно нулю. В этом случае равны нулю все остальные биты мантиссы и кода порядка (что дает минимальный порядок, равный -128). Таким образом, представление нуля с плавающей запятой эквивалентно двум словам с нулями в целочисленной арифметике, что оказывается удобным в ряде случаев. Следует отметить, что в ЭВМ автоматически приравниваются к нулю все результаты, имеющие порядок -128, независимо от значения

Рис. 3.13. В шестнадцатиразрядных ЭВМ числа с плавающей запятой представляются в виде двух машинных слов.



а $3/8 = 1/4 + 1/8 = 0,0110000\ldots_2$

б $0{,}011_2 = 0{,}11_2 \cdot 2^{-1}$

в $-1 \Rightarrow 01111111$ (СМЕЩЕНИЕ 2^7)

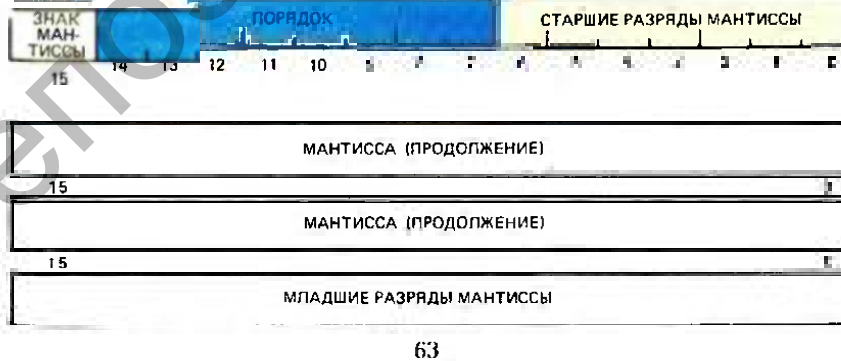
Рис. 3.14. Этапы перевода десятичной дроби $^3/_8$ в формат с плавающей запятой: десятичная дробь записывается в двоичном виде *(а)*; полученный результат нормализуется путем сдвига запятой на нужное число позиций, а число сдвигов дает порядок *(б)*; порядок записывается в коде со смещением *(в)*. На последнем этапе *(г)* удаляется первый бит мантиссы и выписывается полное представление дроби в виде двух машинных слов.

мантисс. Правила перевода десятичных дробей в формат с плавающей запятой иллюстрируются рисунком 3.14.

Описанный формат дает числа в диапазоне примерно от 10^{-38} до 10^{39} с точностью около семи десятичных разрядов. Когда такой точности не хватает, используется формат удвоенной точности, отличающийся тем, что для записи мантиссы отводится два дополнительных машинных слова (рис. 3.15). Это дает точность около пятнадцати десятичных разрядов в том же диапазоне порядков.

Вместо термина «плавающая запятая» часто используется *«плавающая точка»*. Это означает одно и то же. Дело в том, что в записи чисел на языках программирования высокого уровня вместо десятичной (двоичной) запятой используется символ точки. Удобство такого подхода очевидно всем, кто сталкивался с необходимостью записи нескольких подряд десятичных дробей.

Рис. 3.15. Числа с плавающей запятой удвоенной точности записываются четырьмя машинными словами.



3.15. Арифметика чисел с плавающей запятой и ее особенности

Выполнение арифметических операций над числами с плавающей запятой в разных ЭВМ производится по-разному. Процессоры простых микро-ЭВМ не имеют команд, позволяющих непосредственно оперировать числами в этом формате. С этой целью приходится составлять специальные программы, использующие команды целочисленных арифметических действий. Системы команд сложных ЭВМ включают *команды арифметики с плавающей запятой,* которые благодаря аппаратной реализации выполняются значительно быстрее. Как компромиссный вариант начальная конфигурация микро-ЭВМ, рассчитанная на целочисленные действия, может пополняться приобретаемым отдельно *процессором плавающей запятой.*

В любом случае алгоритмы арифметических действий над числами в плавающем формате одинаковы, разница состоит в их программной или аппаратной реализации. Так, для выполнения умножения нужно перемножить мантиссы чисел и сложить их порядки. Умножение 24-разрядных мантисс проще всего выполнить в виде 24-кратного цикла из операций сдвига и сложения (см. 3.6) с определением знака результата по хорошо известному правилу знаков (для умножения и деления). Именно по этой причине мантиссу удобнее представлять не в дополнительном коде, а в представлении «знак-величина» (см. 3.12).

Так как нормализованные мантиссы операндов находятся в диапазоне от 0,5 до 1, то результат их умножения лежит в границах от 0,25 до 1 и может оказаться *денормализованным.* Для его нормализации можно сдвинуть мантиссу на один разряд влево и одновременно уменьшить на единицу порядок результата.

В отличие от умножения и деления при выполнении сложения и вычитания основную сложность составляют не собственно эти действия, а процедуры *выравнивания.* Как и в «ручных» операциях над числами в экспоненциальном представлении в первую очередь необходимо произвести сравнение порядков. Если они различны, то мантисса операнда с меньшим порядком сдвигается вправо, а порядок, соответственно, увеличивается. Выравнивание производится до тех пор, пока порядки обоих операндов не сравняются. После этого производится сложение или вычитание мантисс. Наконец, как и для умножения, может понадобиться нормализация результата. В процедурах сравнения порядков представление дополнительным кодом не очень удобно, поэтому-то порядок чисел с плавающей запятой обычно записывается в коде со смещением, упрощающем алгоритмы выравнивания.

3.16. Погрешности машинной арифметики

Как и для целочисленной арифметики, в операциях с плавающей запятой возможно негативное влияние переполнения. Так, двоичная запятая результата умножения может оказаться правее младшего бита воображаемого 128-битового представления слова с плавающей запятой (см. 3.13). Эта ситуация почти аналогична рассмотренному в 3.6 переполнению для целочисленных операций. Отличие состоит в том, что переполнение в операциях с плавающей точкой обнаруживается не по изменению состояния ССП, а по недопустимо большому значению порядка. Операции над нормализованными мантиссами в принципе не могут вызвать переполнения с плавающей точкой, даже если установится бит переполнения ССП. Так, при сложении двух положительных мантисс может возникнуть перенос в знаковый разряд, который на первый взгляд искажает результат (он кажется отрицательным). На самом деле все становится на свои места в результате нормализации.

Результат вычислений с плавающей запятой может не вписаться в разрядную сетку ЭВМ и по другой причине: когда запятая результата находится левее крайней слева цифры условного 128-битового представления. Это происходит, когда порядок результата — недопустимо большое отрицательное число. Такое «переполнение наоборот» называется *исчезновением порядка:* результат становится слишком малым для его представления в стандартном, нормализованном виде. Естественная реакция ЭВМ в этих ситуациях состоит в интерпретации результата как «чистого нуля».

Очевидно, исчезновение порядка приводит к погрешностям в вычислениях. Неточности в операциях с плавающей запятой могут возникнуть и по другим причинам. В их основе — использование укороченных форм записи чисел: так из 128 бит условного слова с плавающей запятой в мантиссе остается только 24 разряда. Поэтому при вычитании двух почти равных чисел с одинаковыми порядками и мантиссами, отличающимися, скажем, в n младших битах, $24 - n$ старших бита результата окажутся равными нулю. После нормализации такого результата лишь n битов могут оказаться значащими. Вообще, каждая процедура выравнивания приводит к потере точности: так, при сдвиге вправо теряются значащие младшие биты. Еще один источник ошибок кроется в невозможности представить конечным машинным словом бесконечные периодические дроби.

Все перечисленные обстоятельства приводят к тому, что при выполнении громоздких вычислений, каждое из которых опирается на предыдущий результат, ошибки накапливаются и могут достигнуть значительной величины. В некоторой мере эти ошибки удается уменьшить за счет округления промежуточных результатов. Так, при сдвиге мантиссы вправо можно не просто отбрасывать младшие биты, а корректировать с учетом их значения оставшийся результат. Это можно сделать, например, путем прибавления еди-

ницы к результату, если старший из отбрасываемых битов равен единице.

Однако в процедуре округления существуют свои «ловушки» и иногда точность результата с применением округления может оказаться ниже. По этой причине существует множество алгоритмов округления. К сожалению, алгоритмы, приемлемые в одних ситуациях, плохо работают в других. Поэтому при решении задач на ЭВМ не следует удивляться иногда получаемым результатам типа $2 \cdot 2 = 3{,}999999$. Зная особенности используемых алгоритмов округления, можно либо устранить подобные аномалии, либо просто привыкнуть к ним.

3.17. ЭВМ и азбука Морзе (кодирование алфавитно-цифровых символов)

Попробуем выяснить, что происходит, когда на дисплее, связанном с ЭВМ, появляется изображение очередной буквы или цифры. Думается, что основные особенности этого процесса читателю уже понятны. В памяти машины должны храниться двоичные коды выводимого сообщения. Код текущего символа передается в дисплей, где специальными схемами обеспечивается его прием, распознавание и вывод изображения соответствующего символа в нужном месте экрана. В режиме ввода информации нажатие одной из клавиш (или комбинации двух клавиш) приводит к передаче и запоминанию в ЭВМ соответствующего кода. Рассмотрим правила получения двоичных кодов алфавитно-цифровых символов и расшифровки этих кодов.

История этого вопроса уходит в тридцатые годы прошлого столетия, когда американский изобретатель Сэмюэль Морзе начал разработку первого электрического телеграфного аппарата. Буквам латинского алфавита Морзе поставил в соответствие комбинации точек и тире. Это делалось очень остроумным способом: комбинация была тем короче, чем чаще встречалась соответствующая буква в различных сообщениях. Так, буква Е, самая распространенная в английском языке, обозначалась одной точкой. Самыми длинными комбинациями кодировались редко встречающиеся буквы*. Такой подход явился предвестником эффективного (с точки зрения компактности и меньшей подверженности влиянию помех при передаче) кодирования, методы которого были созданы в середине XX века**.

---

* Относительные частоты букв английского языка были определены Морзе не менее остроумно: путем подсчета литер в ячейках типографской кассы для набора текстов.

** Эти методы обосновываются в математической теории связи (другое название — теория информации), созданной в 1948 году выдающимся американским ученым Клодом Шенноном. Основы этой теории увлекательно изложены в научно-популярной книге Дж. Пирса «Символы, сигналы, шумы». Любопытно, что в начале своей работы Морзе применял еще более эффективный с точки зрения теории информации способ кодирования, при котором элементарные комбинации точек и тире соответствовали не отдельным буквам, а целым словам.

Если в *азбуке Морзе* обозначать тире единицей, а точку — нулем, то любой символ может быть выражен двоичным числом длиной максимум в пять бит (Морзе использовал $2^5 = 32$ различных комбинаций точек и тире, что хватало для обозначения латинских букв). Фактически в аппаратах Морзе использовалась троичная система: короткая посылка тока — точка, длинная посылка — тире, отсутствие тока — символ разделителя информации. Третий символ необходим для разделения как самих точек и тире (короткая пауза, с длительностью как у точки), отдельных букв (пауза той же длительности, что у тире), так и слов (пауза вдвое большей длительности).

В обычных телеграфных аппаратах, применяемых и по сей день, чаще используют двоичный *код Бодо*. В этом коде все буквы передаются комбинациями непременно из пяти сигналов, а сами сигналы представляют собой посылки тока или паузы одинаковой длительности. Специальные разделители не нужны: всегда известно, что следующая буква начнется спустя ровно пять сигналов, а для обозначения промежутков между словами можно использовать специальный символ пробела, кодируемый как одна из букв.

Основной недостаток рассмотренных кодов — малое число комбинаций, которых не хватает даже для всех букв русского алфавита, не говоря уже про цифры и знаки препинания: для их обозначения приходится идти на всевозможные ухищрения. Так, всем известно, что точка в телеграмме обозначается как «тчк».

Интересно, что увеличить число представляемых символов можно и без напрашивающегося перехода к кодам длиной более чем в 5 бит. При этом используется идея, аналогичная переключению верхнего и нижнего регистров в пишущих машинках. Вспомним, что реальное число печатаемых ими символов вдвое больше, чем имеющихся клавиш. Так, в зависимости от положения переключателя регистров нажатие одной и той же клавиши может привести к печати заглавной или прописной буквы. Чтобы реализовать эту идею при передаче информации, два из 32 пятибитных кодов следует использовать как сигналы перевода регистра. Таким образом обеспечивается передача почти удвоенного числа $(32 \cdot 2 - 2)$ символов, которые могут обозначать не только буквы, десятичные цифры и знаки пунктуации, но и исполнять функции контроля и управления аппаратурой связи. Недостаток такого подхода — необходимость надежного запоминания в принимающем устройстве последнего кода переключения. Тем не менее пятибитные коды до сих пор используются в технике связи, они же применялись для работы с алфавитно-цифровой информацией и в первых ЭВМ.

3.18. Стандартные алфавитно-цифровые коды для ЭВМ

В современной вычислительной технике наиболее употребимы семи- и восьмибитные коды, принятые как международные стандарты. Рассмотрим семибитную систему кодов в отечественном стандарте *КОИ-7* (код для обмена информацией, семибитный). Любой код этой системы удобно разделить на два поля, причем первые три бита определяют номер группы символов (зоны), а остальные — номер в зоне. Тогда, в зависимости от значения битов в этих полях, все 128 возможных кодов можно представить в компактном виде (табл. 3.2).

Таблица 3.2

$a_4a_5a_6$ (зона) / $a_3a_2a_1a_0$ (номер в зоне)	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0 0 0 0	NUL	DLE	SP	0	@	P	Ю	П
0 0 0 1	SOH	DC1	!	1	A	Q	А	Я
0 0 1 0	STX	DC2	"	2	B	R	Б	Р
0 0 1 1	ETX	DC3	#	3	C	S	Ц	С
0 1 0 0	EOT	DC4	¤	4	D	T	Д	Т
0 1 0 1	ENQ	NAK	%	5	E	U	Е	У
0 1 1 0	ACK	SYN	&	6	F	V	Ф	Ж
0 1 1 1	BEL	ETB	'	7	G	W	Г	В
1 0 0 0	BS	CAN	(	8	H	X	Х	Ь
1 0 0 1	HT	EM	)	9	I	Y	И	Ы
1 0 1 0	LF	SUB	*	:	J	Z	Й	З
1 0 1 1	VT	ESC	+	;	K	[	К	Ш
1 1 0 0	FF	FS	,	<	L	\	Л	Э
1 1 0 1	CR	GS	—	=	M	]	М	Щ
1 1 1 0	SO	RS	.	>	N	↑	Н	Ч
1 1 1 1	SI	US	/	?	O	—	О	DEL

Наиболее употребимые в вычислительной технике спецсимволы означают: BEL — звонок, CR — возврат каретки (ВК), LF — перевод строки (ПС), DEL — вычеркивание, т. е. забой (ЗБ). Каждый символ в таблице находится на пересечении столбца, определяемого тремя старшими битами кода, и строки, соответствующей четырем младшим битам. Так, код $123_8 = 101\ 0011_2$

означает букву S, а символу CR отвечает код $0001101_2 = 015_8$.

Основные особенности приведенной системы кодирования:

1. *Спецсимволы* (символы управления), расположенные в зонах 0 и 1, в основном предназначены для управления работой устройств, подключаемых к ЭВМ. В устройствах вывода информации, например в печатающих, поступление спецсимвола приводит к выполнению конкретного действия типа перевода каретки и не сопровождается печатью какой-либо информации. Для получения кодов спецсимволов в устройствах ввода используют либо специальные клавиши (возврат каретки, перевод строки), либо комбинации из двух клавиш, что упрощает построение клавиатуры. Обычно одновременно нажимаются клавиша, соответствующая латинской букве строки таблицы 3.2, в которой расположен нужный спецсимвол, и клавиша, предназначенная для получения символов управления (СУ). Так, символ BEL получается одновременным нажатием клавиш СУ и G. (В литературе такую комбинацию принято обозначать СУ/G.)

2. Коды десятичных цифр можно получить, приписывая слева комбинацию 011 к четырехразрядной записи этих цифр в двоичной системе счисления. (В восьмеричной системе к такому же результату приведет сложение восьмеричной записи цифры с числом 60_8.)

3. Символы зон 2—7, за исключением спецсимвола DEL, печатаются или отображаются на устройствах вывода. Символ *DEL (забой)* используется для устранения ошибок. Если ошибка допущена при выводе на носитель информации, с которого можно стереть предыдущую информацию, например на экран дисплея, то подача DEL приводит к стиранию последнего отображаемого символа. На носителях, с которых нельзя стереть информацию, например на перфоленте, ошибочный символ можно устранить пробивками поперек ленты во всех позициях, что соответствует коду DEL (1111111_2).

4. Если в поле зоны средний бит игнорируется, то коды латинских и русских букв, лежащие в одной строке (табл. 3.2), становятся неразличимыми. Иногда это позволяет обойтись устройством вывода, рассчитанным на буквы единственного алфавита (русского или латинского). Из ЭВМ в это устройство могут поступать коды любых букв, но выводиться будут буквы только одного алфавита.

5. Последовательность кодов букв латинского алфавита образует возрастающий ряд двоичных чисел. Это часто используется в программах, например, когда нужно расположить список имен в алфавитном порядке.

6. В англоязычных странах применяется *код ASCII* (см. Список сокращений), в котором зоны 6 и 7 таблицы 3.2 используются для кодирования строчных букв латинского алфавита.

В коде *КОИ-8* для представления каждого символа отводится 8 бит (байт), что позволяет расширить набор символов строчными буквами русского и латинского алфавитов и др. знаками.

ЛОГИКА, МАТЕМАТИКА И ЦИФРОВЫЕ СХЕМЫ

Логика — наука древняя. Ее основоположником считают Аристотеля, жившего в 384—322 годах до н. э.

Ко времени зарождения логики математика уже прошла значительный путь развития. Достаточно вспомнить, что в VI веке до н. э. жил древнегреческий мыслитель Пифагор Самосский. И хотя легенды о нем трудно отделить от действительности, но достоверно известно, что с его именем связано: введение доказательств в геометрию, создание учения о подобии, построение некоторых правильных многоугольников и др. Пифагор доказал в общем виде теорему о сторонах прямоугольного треугольника, носящую его имя, хотя, возможно, она была известна еще ранее.

Логика и математика, достигшие расцвета еще в глубокой древности, имеют принципиально неограниченную область применения: все виды доказательств основаны на законах логики, все количественные соотношения подвластны математике.

В средневековье развитие математики, и особенно логики, замедлилось, потому что новые логические идеи нередко входили в противоречие с формами мышления, основанными на преклонении перед авторитетом церкви. Оживление началось в XVI веке, а уже в XVII—XVIII веках появились предложения о «математизации логики».

Французский философ и математик Рене Декарт (1596—1650) считал, что человеческий разум может постигнуть истину, если будет исходить из достоверных положений, сводить сложные идеи к простым, переходить от известного и доказанного к неизвестному, избегая каких-либо пропусков в логических звеньях исследований. Фактически Декарт рекомендовал науке о мышле-

нии — логике руководствоваться общепринятыми в математике принципами.

Немецкий философ и математик Готфрид Вильгельм Лейбниц предлагает использовать в логике математическую символику и впервые высказывает мысль о возможности применения двоичной системы счисления в вычислительной математике. Но этим идеям Лейбница суждено было получить дальнейшее развитие лишь в середине XIX века в трудах Джорджа Буля (1815—1864), именем которого назван раздел математики — булева алгебра, а также в трудах Огастена де Моргана (1806—1871), Уильяма Стенли Джевонса (1835—1882), Платона Сергеевича Порецкого (1846—1907), Чарлза Сандерса Пирса (1839—1914) и др.

Так зародилась новая наука — математическая логика.

4.1. Логика формальная и математическая

Появлению и развитию математической логики способствовало стремление найти строгие правила обоснования и доказательства новых положений в науке. По мнению некоторых ее создателей математическая логика должна была стать математическим аппаратом той части обычной логики, которую принято называть формальной.

Как и математика, формальная логика следует строгим правилам и не вникает в сущность анализируемых суждений. Одна из ее задач — установление формальных правил получения новых суждений из исходных, истинность которых не подвергается сомнению.

Логические значения высказываний обозначим следующим образом: истинность — цифрой 1, ложность — цифрой 0.

Рассмотрим один из видов сложных высказываний, название которых происходит от латинского *conjunctio* — соединяю. По определению *конъюнктивным* называется сложное высказывание, которое истинно тогда и только тогда, когда истинны все входящие в него высказывания. Пусть число исходных высказываний будет минимальным, т. е. равным двум. Обозначим эти высказывания буквами *B* и *A*. Все возможные наборы их значений и итогового сложного высказывания *Y* сведены в таблицу 4.1 или равнозначную ей таблицу 4.2.

Таблица 4.1

B	*A*	*Y*
ложно	ложно	ложно
ложно	истинно	ложно
истинно	ложно	ложно
истинно	истинно	истинно

Таблица 4.2

B	*A*	*Y*
0	0	0
0	1	0
1	0	0
1	1	1

На первый взгляд вторая таблица не имеет никаких преимуществ перед первой. Но давайте посмотрим, какую из таблиц легче

запомнить. Первую нужно просто-напросто «зазубрить». Вторую же можно запомнить, руководствуясь простыми правилами.

Какие же в ней закономерности? Во-первых, комбинации 0 и 1 в первых двух колонках (B, A) образуют натуральный ряд 2-разрядных двоичных чисел: 00, 01, 10, 11, т. е. в десятичной системе 0, 1, 2, 3. Во-вторых, каждая цифра в колонке Y есть результат обычного математического действия умножения значений B и A в соответствующей этой цифре строке.

Таким образом, чтобы восстановить в памяти правила определения истинности сложного конъюнктивного суждения, достаточно уметь умножать нули и единицы во всех возможных сочетаниях или, что то же самое, знать таблицу умножения одноразрядных двоичных чисел:

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

Еще один вид сложных суждений обязан своим названием латинскому *disjunctio* — разобщение, различие. В формальной логике неисключающим дизъюнктивным суждением называется сложное суждение, которое истинно тогда и только тогда, когда истинно хотя бы одно из входящих в него суждений. Как и в предыдущем случае, составим две таблицы, соответствующие сформулированному определению.

Таблица 4.3

B	A	Y
ложно	ложно	ложно
ложно	истинно	истинно
истинно	ложно	истинно
истинно	истинно	истинно

Таблица 4.4

B	A	Y
0	0	0
0	1	1
1	0	1
1	1	1

Математическую операцию, которая может быть пояснена таблицей 4.3, называют в *булевой алгебре* дизъюнкцией, или логическим суммированием. Применяя для ее обозначения знак «+», можем записать:

$$Y = B + A.$$

Еще раз напомним, что это не обычное, а логическое суммирование, поэтому

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

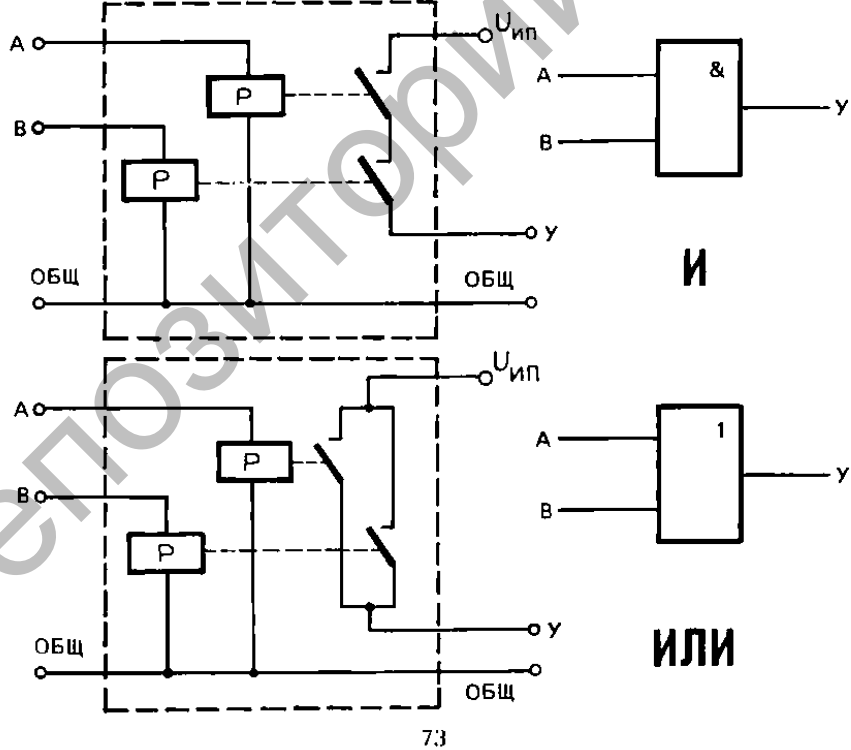
4.2. Схемы И, ИЛИ и НЕ

Начнем с бытовых аналогий. Если дверь закрывается на два (или три) замка, она сконструирована как своего рода схема И. В помещении с такой дверью можно войти, открыв **и** первый, **и** второй (**и** третий, если он есть) замок.

Но не торопитесь с выводами. Эту же дверь можно считать и схемой ИЛИ, так как она будет закрыта в случаях, когда закрыт **или** первый, **или** второй, **или** третий, **или** любая пара замков, **или** все три.

Чтобы избежать путаницы, условимся, как это принято в вычислительной технике, считать схемами И, ИЛИ только такие устройства, у которых сигналы на входах и выходах имеют одинаковую физическую природу. Обычно за 0 принимается низкий, а за единицу высокий уровень напряжения. В принципе схемы И и ИЛИ могут быть электрическими, механическими, пневматическими и иными, важно лишь, повторяем, чтобы 0 и 1 были идентичными как на входе, так и на выходе.

Рис. 4.1. Схемы И и ИЛИ на основе реле. В общем случае каждый вход и выход представляет собой пару проводов. Внешние соединения упрощаются, если один из проводов каждой такой пары — общий. Условные обозначения логических схем одинаковы независимо от того, из каких элементов они построены. Символ & обозначает «и».



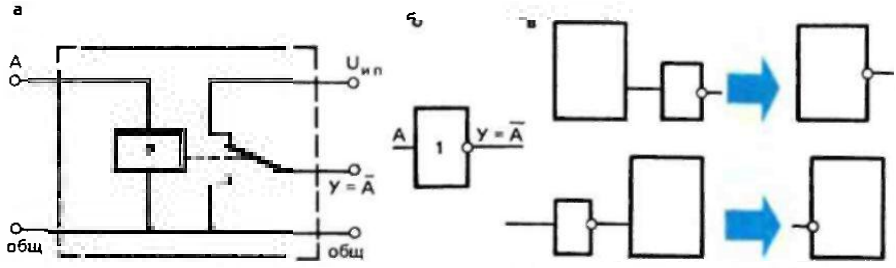


Рис. 4.2. **В схеме НЕ на основе реле *(а)* при замыкании входа *А* с общим проводом выход *У* отключается от этого провода. Условное обозначение элемента НЕ *(б)*. Обозначения упрощаются, если инвертирование (отрицание) обозначается кружочком на выходе или входе логической схемы *(в)*.**

Теперь позволительно перейти и к строгим определениям.

1. **Схема И** — устройство, имеющее два или несколько входов и один выход. Сигнал 1 появляется на выходе только в том случае, когда сигналы на всех входах 1.

2. **Схема ИЛИ** — устройство с одним выходом и двумя или несколькими входами. Логическая 1 появляется на выходе, когда хотя бы на одном входе есть 1.

Однотипность сигналов на входах и выходах позволяет подключать выходы одних схем ко входам других, причем к одному выходу, если это необходимо, могут быть присоединены входы нескольких схем.

На рисунке 4.1 показано, как можно сделать схемы И и ИЛИ. Здесь же приведены условные обозначения схем, которыми удобно пользоваться независимо от того, каково их конкретное устройство.

Обратите внимание на то, что работа схемы И поясняется операцией логического умножения и таблицей 4.2, а схемы ИЛИ — операцией логического суммирования и таблицей 4.4.

3. Рассмотрим **схему НЕ.** Она имеет один вход и один выход и превращает 0 в 1, а 1 в 0. Такое преобразование, называемое отрицанием, или инверсией, обозначается черточкой над преобразуемой переменой: $Y = \bar{A}$. В отличие от схем И и ИЛИ схему НЕ невозможно построить без активного элемента, т. е. без устройства, способного увеличивать мощность электрического сигнала — транзистора, электронной лампы или электромагнитного реле.

На рисунке 4.2 показано, как можно сделать схему НЕ на основе реле, если, как и ранее, за 0 принят низкий уровень напряжения, а за 1 — высокий.

4.3. Поиграем в «крестики-нолики»

Предлагаемое занятие действительно напоминает игру в крестики-нолики, только вместо крестика будет ставиться единица.

Поле для игры или диаграмма — четыре клеточки, образующие квадрат. Правила игры — в каждую клеточку нужно обяза-

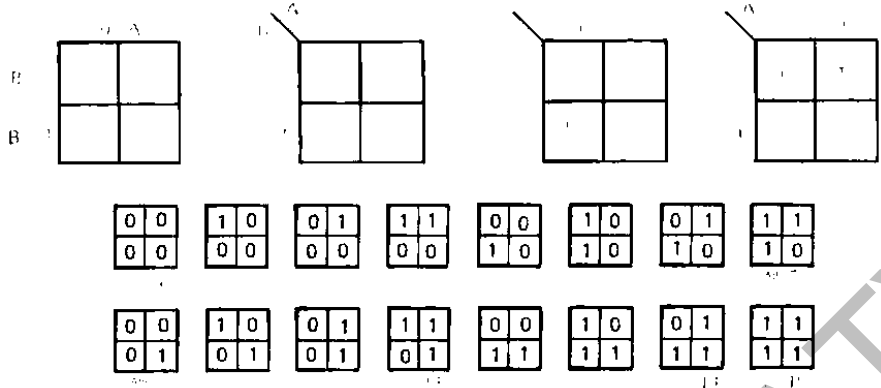


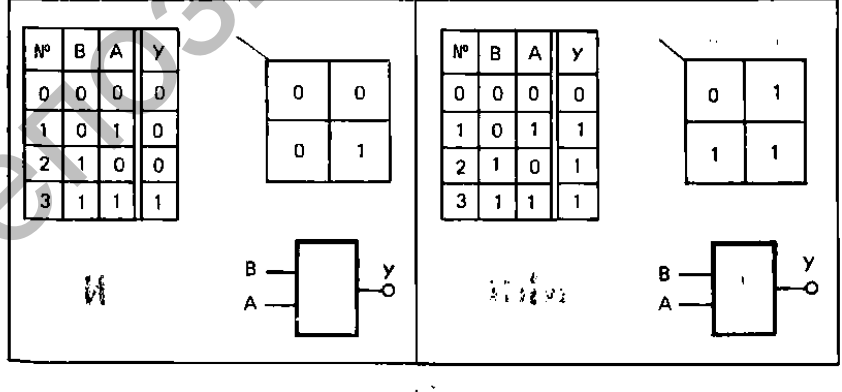
Рис. 4.2. Четырехклеточные диаграммы--одна из форм описания работы двухвходовых цифровых схем. Сигналам на входе *B* соответствуют строки, а сигналам на входе *A* — столбцы диаграмм. В клеточках, каждая из которых соответствует строго определенному набору сигналов *BA*, записывают сигналы на выходе системы (№ № 0...15).

тельно вписать 0 или 1. Выигрывает тот, кто предложит больше отличающихся друг от друга вариантов заполнения диаграммы.

Если играющему известна двоичная система счисления, то он никогда не проиграет. Более того, он с уверенностью может заявить, что существует ровно 16 вариантов различного заполнения такой четырехклеточной диаграммы, так как предложенная задача равносильна вопросу: «Сколько существует различных четырехразрядных двоичных чисел?»

Все возможные варианты заполнения четырехклеточной диаграммы показаны на рисунке 4.3. Они имеют непосредственное отношение к логическим цифровым схемам. Чтобы убедиться в этом, пронумеруем столбцы и строки цифрами 0 и 1.

Рис. 4.3. Различные формы описания работы схем И и ИЛИ.



На рисунке 4.4 воспроизведены парами таблица 4.2 и диаграмма № 8 с рисунка 4.3, а также таблица 4.4 и диаграмма № 14. Нетрудно заметить, что эти пары полностью равнозначны, т. е. схему И можно описать диаграммой № 8, а схему ИЛИ — диаграммой № 14.

Действительно, каждой клеточке диаграммы (как и каждой строке таблицы) соответствует вполне определенный набор значений B и A, а цифра в этой же клеточке указывает, какой сигнал получается на выходе схемы при таком наборе.

Напрашивается вывод, что рассмотренные схемы И и ИЛИ — лишь отдельные представители семейства из 16 возможных двухвходовых схем. Не все эти схемы в полном смысле двухвходовые. На устройства с № 0 и № 15 вообще не оказывает влияния ни сигнал A, ни сигнал B, устройства с № 3, 5, 10, 12 реагируют на изменение сигнала только на одном из входов. Подробно об этом расскажем позже. Теперь же ознакомимся с основами той алгебры, которая описывает эти устройства.

4.4. Вся алгебра в одной функции!

В алгебре буквенные обозначения заменяют произвольные числа — положительные, отрицательные, дробные, мнимые и др. В булевой алгебре буквами обозначены переменные, которые могут принимать только два значения — 0 или 1. Функции от любого количества таких переменных, называемые *переключательными*, или *булевыми*, также могут принимать значение только 0 или 1. Поэтому существует лишь строго определенное количество функций с заданным числом переменных. Его нетрудно определить.

Сначала установим, сколько может быть наборов переменных. Если переменная одна, т. е. функция имеет вид $\Phi(A)$, то таких наборов два: $A=0$ и $A=1$. При двух переменных, т. е. для функции $\Phi(B, A)$, таких наборов значений BA четыре: 00, 01, 10, 11. У функции трех переменных $\Phi(C,B,A)$ наборов CBA восемь: 000, 001, 010, 011, 100, 101, 110, 111. При числе переменных n количество наборов $m=2^n$, и они представляют собой натуральный ряд n-разрядных двоичных чисел от 0 до 2^n-1.

Переключательная функция при каждом наборе переменных принимает значение 0 или 1, поэтому отличающихся друг от друга функций может быть ровно столько, сколько существует различных комбинаций из 2^n единиц и нолей. Таких комбинаций 2^m, и они представляют собой последовательность m-разрядных двоичных чисел от 0 до 2^m-1. Таким образом, общее количество функций n переменных равно

$$2^m=(2)^{2n}$$

Не стоит расстраиваться из-за такого быстрого роста числа функций с увеличением n: знакомиться с ними нет никакой необходимости хотя бы потому, что **функция любого количества**

переменных может быть выражена через функции только двух переменных. Делается это при помощи очень простых приемов. Во-первых, путем подстановки на место переменных других булевых функций (а это можно делать, так как и функция и переменные принимают только два значения — 0 и 1). Во-вторых, перестановкой переменных местами («перенумерацией»). Оба эти приема объединяются общим названием — *суперпозиция.*

В таблицы 4.5 и 4.6 сведены соответственно все функции одной и двух переменных. Однако многие функции таблицы никак не назовешь зависящими от двух переменных. Некоторые из них вообще не зависят ни от одной переменной (Φ_0, Φ_{15}), так как при любых значениях переменных их значения остаются неизменными.

Таблица 4.5

Функция	Значения функции		Название	Обозначение
	при $A = 1$	при $A = 0$		
$\Phi_0(A)$	0	0	константа 0	0
$\Phi_1(A)$	0	1	отрицание (инверсия) A	$\overline{A}$
$\Phi_2(A)$	1	0	переменная A	A
$\Phi_3(A)$	1	1	константа 1	1

Таблица 4.6

Функция	Значения при		Название
	$B=$	1 1 0 0	
	$A=$	1 0 1 0	
$\Phi_0(B, A)$	0 0 0 0		константа 0
$\Phi_1(B, A)$	0 0 0 1		отрицание дизъюнкции
$\Phi_2(B, A)$	0 0 1 0		
$\Phi_3(B, A)$	0 0 1 1		отрицание B (инверсия)
$\Phi_4(B, A)$	0 1 0 0		
$\Phi_5(B, A)$	0 1 0 1		отрицание A (инверсия)
$\Phi_6(B, A)$	0 1 1 0		неравнозначность (сумма по модулю 2)

Функция	Значения при		Название
	$B=$	1 1 0 0	
	$A=$	1 0 1 0	
$\Phi_7(B, A)$		0 1 1 1	отрицание конъюнкции
$\Phi_8(B, A)$		1 0 0 0	конъюнкция
$\Phi_9(B, A)$		1 0 0 1	равнозначность
$\Phi_{10}(B, A)$		1 0 1 0	переменная A
$\Phi_{11}(B, A)$		1 0 1 1	
$\Phi_{12}(B, A)$		1 1 0 0	переменная B
$\Phi_{13}(B, A)$		1 1 0 1	
$\Phi_{14}(B, A)$		1 1 1 0	дизъюнкция
$\Phi_{15}(B, A)$		1 1 1 1	константа 1

Нельзя ли, применяя метод суперпозиции, выразить одни из этих функций через другие, а если можно, то каково минимальное количество функций двух переменных, через которые этим методом выражаются все остальные функции?

Оказывается, что *функционально полный набор булевых функций*, т. е. такой набор, через функции которого методом суперпозиции выражается любая булева функция двух переменных, может содержать всего одну функцию! При желании к ней можно свести всю алгебру логики.

Несколько примеров функционально полных наборов:

1) функции Φ_5 и Φ_8;
2) функции Φ_3 и Φ_{14};
3) одна функция Φ_1;
4) одна функция Φ_7.

Сравнив таблицу 4.6 с рисунком 4.3, легко заметить, что каждая строка таблицы и диаграмма с таким же порядковым номером несут одну и ту же информацию. И та и другая отвечает на вопрос, чему равна функция (сигнал на выходе) при каждом конкретном наборе значений переменных (входных сигналов). Таблица 4.6 в целом эквивалентна набору из 16 диаграмм на рисунке и содержит все сведения о работе любой двухвходовой цифровой схемы. Поэтому логические схемы, для описания которых применяются функции Φ_1 и Φ_7, т. е. *схемы ИЛИ-НЕ* и *И-НЕ*, также универсальны, как и сами функции. Любое логическое и арифметическое устройство любой ЭВМ может быть сделано из логических схем только одного типа, например только двухвходовых схем И-НЕ.

4.5. Как доказать теоремы булевой алгебры

Любое доказательство основано на положениях, считающихся справедливыми. Такие исходные положения не доказываются и называются аксиомами. *Аксиомы булевой алгебры* — это краткая математическая форма записи того, что переменные принимают только два значения — 0 и 1; что операция инверсии (отрицания) превращает 0 в 1 и наоборот и, наконец, что логическое умножение и суммирование выполняют так, как это предписано таблицами 4.2 и 4.4. Вот эти аксиомы:

$$A = 0, \text{ если } A \neq 1; \qquad A = 1, \text{ если } A \neq 0; \tag{4.1}$$
$$A = 0;\ \bar{A} = 1; \qquad A = 1;\ \bar{A} = 0; \tag{4.2}$$
$$A + 0 = A; \qquad A \cdot 1 = A; \tag{4.3}$$
$$A + 1 = 1; \qquad A \cdot 0 = 0; \tag{4.4}$$
$$A + \underline{A} = A; \qquad A \cdot \underline{A} = A; \tag{4.5}$$
$$A + \bar{A} = 1; \qquad A \cdot \bar{A} = 0. \tag{4.6}$$

На них основано доказательство *теорем булевой алгебры*, называемых по традиции законами. С помощью этих законов можно преобразовывать и упрощать выражения. Убедиться в справедливости законов, т. е. фактически доказать теоремы, можно при помощи подстановки на место переменных всех возможных сочетаний 0 и 1.

Закон коммутативности утверждает, что

$$A + B = B + A \text{ и } A \cdot B = B \cdot A \tag{4.7}$$

Его доказательство предлагаем сделать самому читателю. Мы же докажем первый **закон поглощения** переменной:

$$A + A \cdot B = A \text{ и } A(A + B) = A. \tag{4.8}$$

Проверку выполнения этих равенств при всех наборах A и B представим в виде таблицы 4.7.

Таблица 4.7

Наборы		Закон поглощения	
A	B	$A + A \cdot B = A$	$A(A + B) = A$
0	0	$0 + 0 \cdot 0 = 0$	$0(0 + 0) = 0$
0	1	$0 + 0 \cdot 1 = 0$	$0(0 + 1) = 0$
1	0	$1 + 1 \cdot 0 = 1$	$1(1 + 0) = 1$
1	1	$1 + 1 \cdot 1 = 1$	$1(1 + 1) = 1$

Все равенства выполняются, поэтому первый закон поглощения доказан.

Аналогично доказываются и остальные законы.

Второй закон поглощения переменной:

$$A + \bar{A} \cdot B = A + B;\ A(\bar{A} + B) = A \cdot B. \qquad (4.8а)$$

Закон ассоциативности:

$$(A + B) + C = A + (B + C) = A + B + C. \qquad (4.9)$$
$$(A \cdot B) \cdot C = A \cdot (B \cdot C) = A \cdot B \cdot C.$$

Закон дистрибутивности:

$$A + B + C = (A + B) \cdot (A + C). \qquad (4.10)$$
$$A(B + C) = A \cdot B + A \cdot C.$$

Закон инверсии:

$$\bar{A} + \bar{B} = \overline{A \cdot B};\qquad \bar{A} \cdot \bar{B} = \overline{A + B}. \qquad (4.11)$$

Поскольку закон 4.11 не столь очевиден, как предыдущие, проверим его, сведя результаты проверки в таблицу 4.8.

Таблица 4.8

Наборы		Закон инверсии	
A	B	$\overline{A+B}=\bar{A}\cdot\bar{B}$	$\overline{A\cdot B}=\bar{A}+\bar{B}$
0	0	$\overline{0+0}=\bar{0}\cdot\bar{0}$, т. е. $1=1$	$\overline{0\cdot 0}=\bar{0}+\bar{0}$, т. е. $1=1$
0	1	$\overline{0+1}=\bar{0}\cdot\bar{1}$, т. е. $0=0$	$\overline{0\cdot 1}=\bar{0}+\bar{1}$, т. е. $1=1$
1	0	$\overline{1+0}=\bar{1}\cdot\bar{0}$, т. е. $0=0$	$\overline{1\cdot 0}=\bar{1}+\bar{0}$, т. е. $1=1$
1	1	$\overline{1+1}=\bar{1}\cdot\bar{1}$, т. е. $0=0$	$\overline{1\cdot 1}=\bar{1}+\bar{1}$, т. е. $0=0$

4.6. Как синтезируют комбинационные схемы

Если алфавит, применяемый на входе и выходе КС (см. 2.2 и 2.3), чисто двоичный, то каждому двоичному коду на входе КС ставит в соответствие вполне определенный код на выходе или, что то же самое, каждой комбинации нулей и единиц на входе соответствует строго определенная комбинация нулей и единиц на выходе (отсюда и название — комбинационная).

Любую КС, имеющую m выходов, можно заменить набором из m комбинационных схем, каждая из которых имеет только один выход и такое же количество входов, как и исходная КС. Очевидно, для постройки любой КС достаточно научиться составлять (синтезировать) произвольную КС с одним выходом. Именно они будут называться далее просто КС.

Итак, комбинационной будем называть схему, сигнал на выходе которой однозначно определяется комбинацией сигналов на входах. К ним относятся, в частности, схемы И и ИЛИ.

Из двухвходовых схем И и из двухвходовых схем ИЛИ можно создать схемы И и ИЛИ с любым количеством входов. Как это делается — показано на рисунке 4.5. Зная свойства двухвходовых

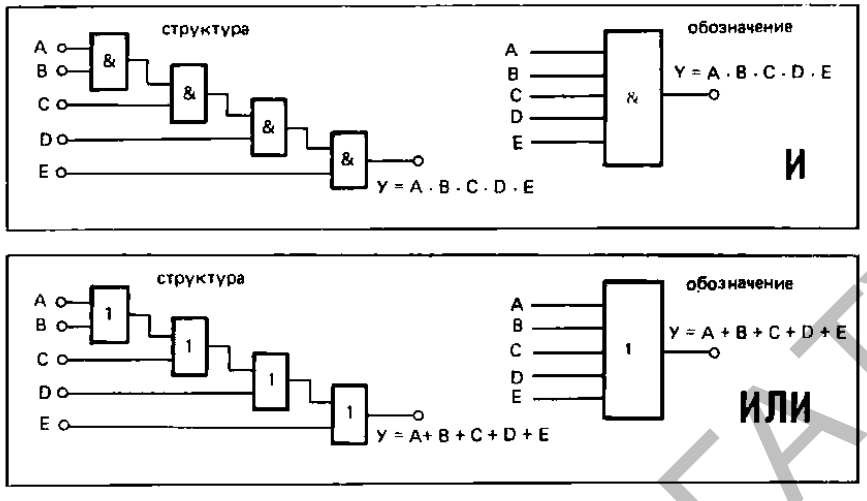


Рис. 4.5. Из двухвходовых схем можно построить многовходовые схемы И и ИЛИ.

схем, легко проверить, что многовходовые схемы И и ИЛИ работают в соответствии с определением, данным в 4.3.

Любую другую комбинационную схему также можно синтезировать путем соединения между собой схем И, ИЛИ и НЕ.

Свойства комбинационной схемы полностью определяет *таблица значений* соответствующей логической функции, в которой указывается, какой сигнал должен быть на выходе при каждом возможном наборе сигналов на входах.

Синтез произвольной КС по заданной таблице значений заключается в получении выражения булевой алгебры, описывающего работу схемы, в упрощении этого выражения *(минимизации)* и, наконец, в составлении самой искомой схемы по минимизированному выражению.

Один из способов такого синтеза основывается на использовании так называемых *простых конъюнкций*. При числе переменных n (т. е. при синтезе n-входовой схемы) число простых конъюнкций равно 2^n.

В таблицах 4.9 показано, как образуются простые конъюнкции для $n = 2$ и $n = 3$. При большем n принцип остается тем же. Данному набору переменных соответствует простая конъюнкция, т. е. произведение всех без исключения переменных, в котором переменные, равные 0, берутся со знаком инверсии, а равные 1 — без него.

Так, например, простая конъюнкция для набора $E = 0$, $D = 1$, $C = 0$, $B = 0$, $A = 1$ есть $Y_9 = \overline{E} \cdot D \cdot \overline{C} \cdot \overline{B} \cdot A$.

Выражение булевой алгебры для описания синтезируемой схемы получают на основании теоремы: **любая булева функция может быть представлена в виде логической суммы простых**

Таблица 4.9

№	B	A	простая конъюнкция
0	0	0	$\bar{B}\cdot\bar{A}$
1	0	1	$\bar{B}\cdot A$
2	1	0	$B\cdot\bar{A}$
3	1	1	$B\cdot A$

$n=2$

№	C	B	A	простая конъюнкция
0	0	0	0	$\bar{C}\cdot\bar{B}\cdot\bar{A}$
1	0	0	1	$\bar{C}\cdot\bar{B}\cdot A$
2	0	1	0	$\bar{C}\cdot B\cdot\bar{A}$
3	0	1	1	$\bar{C}\cdot B\cdot A$
4	1	0	0	$C\cdot\bar{B}\cdot\bar{A}$
5	1	0	1	$C\cdot\bar{B}\cdot A$
6	1	1	0	$C\cdot B\cdot\bar{A}$
7	1	1	1	$C\cdot B\cdot A$

$n=3$

конъюнкций, соответствующих тем наборам переменных, при которых эта функция принимает значение 1.

Воспользуемся теоремой и получим в качестве примера булевы выражения, описывающие некоторые двухвходовые схемы в таблице 4.3. Для схемы № 6 (ИСКЛЮЧАЮЩЕЕ ИЛИ) единица на выходе появляется при наборах: 1) $B=0, A=1$ и 2) $B=1, A=0$, поэтому

$$Y_6=\bar{B}\cdot A+B\cdot\bar{A}. \tag{4.12}$$

Для схемы И-НЕ (№ 7) имеем:

$$Y_7=\bar{B}\cdot\bar{A}+\bar{B}\cdot A+B\cdot\bar{A}. \tag{4.13}$$

4.7. Минимизация

Некоторые выражения, составленные по только что сформулированной теореме, можно упростить, применяя аксиомы и законы булевой алгебры 4.1—4.11. Дело это несложное, но требует аккуратности и внимания, а допущенная при процедуре упрощения неточность приводит к ошибочному результату. Чтобы исключить подобного рода неточности и избавиться от однообразной и малоинтересной работы, упрощения выражений производят при помощи диаграмм минимизации.

Диаграммы минимизации содержат 2^n клеточек, каждая из которых предназначена для одной из простых конъюнкций или, что то же самое, для одного набора значений переменных. В случае $n=2$ диаграмма для минимизации совпадает с принятой в 4.4 (рис. 4.3 и 4.6, *а*) формой представления таблицы значений функции двух переменных. В этой диаграмме любые соседние клетки отличаются лишь появлением или исчезновением знака инверсии над одной переменной, а соответствующие им наборы переменных отличаются лишь одной цифрой. Этот же принцип сохраняется и при большем количестве переменных.

Для $n=3$ диаграмма строится путем пририсовки к диаграмме для двух переменных ее зеркального изображения (рис. 4.6, *б*). Всем клеточкам «старой» части ставится в соответствие новая

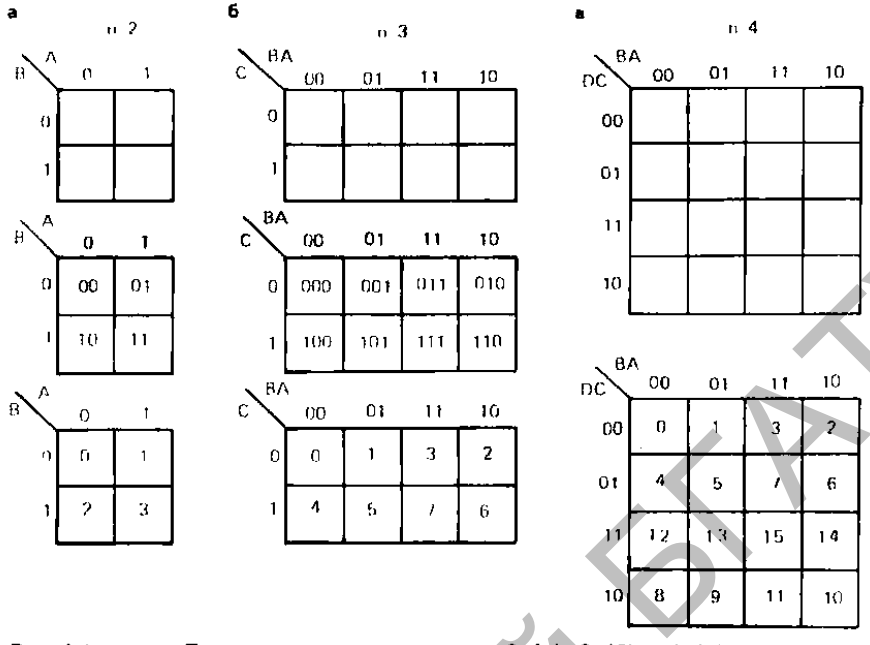


Рис. 4.6. **Диаграммы минимизации для 2 *(а)*, 3 *(б)* и 4 *(в)* переменных.**

переменная, равная 0, «пририсованной» — равная 1. Таким же способом строят диаграммы для любого количества переменных, например для $n = 4$ (рис. 4.6, *в*).

Используя диаграммы, нет необходимости записывать булево выражение искомой комбинационной схемы. Достаточно лишь занести 1 в клеточки, соответствующие наборам переменных, при которых на выходе комбинационной схемы должен быть сигнал 1 (вспомните «крестики-нолики»!).

Если занесенные единицы образуют прямоугольник из 2^n клеточек, то логическая сумма соответствующих им простых конъюнкций может быть заменена укороченной конъюнкцией, содержащей лишь те переменные, значения которых одинаковы для всех клеточек данного прямоугольника. Для этого такие массивы из 2, 4, 8 и т. д. клеточек обводятся контурами. Искомое минимизированное булево выражение есть сумма конъюнкций контуров, охватывающих все без исключения единицы. При этом крайние правый и левый столбцы диаграммы и верхняя и нижняя ее строки считаются соседними, а контуры могут перекрываться. Чем объясняется такая процедура минимизации, поясним на примере выражения (4.13). Сначала упростим его обычным методом, используя формулы (4.5), (4.7), (4.8) и (4.10):

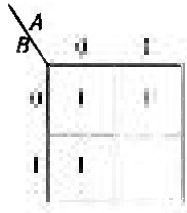
$$Y_7 = \bar{B}\cdot\bar{A} + \bar{B}\cdot A + B\cdot\bar{A} = \bar{B}\cdot\bar{A} + \bar{B}\cdot A + \bar{B}\cdot\bar{A} + B\cdot\bar{A} =$$
$$= (\bar{B}\cdot\bar{A} + \bar{B}\cdot A) + (\bar{B}\cdot\bar{A} + B\cdot\bar{A}) =$$
$$\bar{B}(\bar{A} + A) + \bar{A}(\bar{B} + B) = \bar{B} + \bar{A}. \qquad (4.14)$$

Теперь проделаем то же самое, используя диаграмму минимизации для двух переменных (табл. 4.10). Укороченная конъюнкция, получающаяся в результате оконтуривания клеточек $\bar{B} \cdot A$ и $\bar{B} \cdot \bar{A}$, есть просто $\bar{B}$, а укороченная конъюнкция от контура с клеточками $\bar{B} \cdot \bar{A}$ и $B \cdot \bar{A}$ есть просто $\bar{A}$. Таким образом, сразу же можно записать

$$Y_7 = \bar{B} + \bar{A},$$

что совпадает с (4.14). Приведем еще два примера составления и минимизации выражений, описывающих действие схем, заданных таблицами значений (табл. 4.11, 4.12).

Таблица 4.10



или

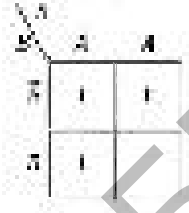


Таблица 4.11

№	C B A	Y
0	0 0 0	1
1	0 0 1	0
2	0 1 0	1
3	0 1 1	0
4	1 0 0	1
5	1 0 1	0
6	1 1 0	1
7	1 1 1	1

$BA \quad Y = \bar{A} + C \cdot B$

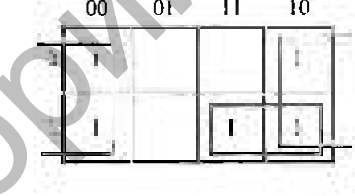
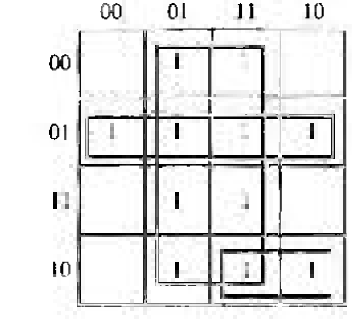


Таблица 4.12

№	D C B A	Y
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	0
3	0 0 1 1	1
4	0 1 0 0	1
5	0 1 0 1	1
6	0 1 1 0	1
7	0 1 1 1	1
8	1 0 0 0	0
9	1 0 0 1	1
10	1 0 1 0	1
11	1 0 1 1	1
12	1 1 0 0	0
13	1 1 0 1	1
14	1 1 1 0	0
15	1 1 1 1	1

$BA \quad Y = A + \bar{D} \cdot C + D \cdot \bar{C} \cdot B$

DC

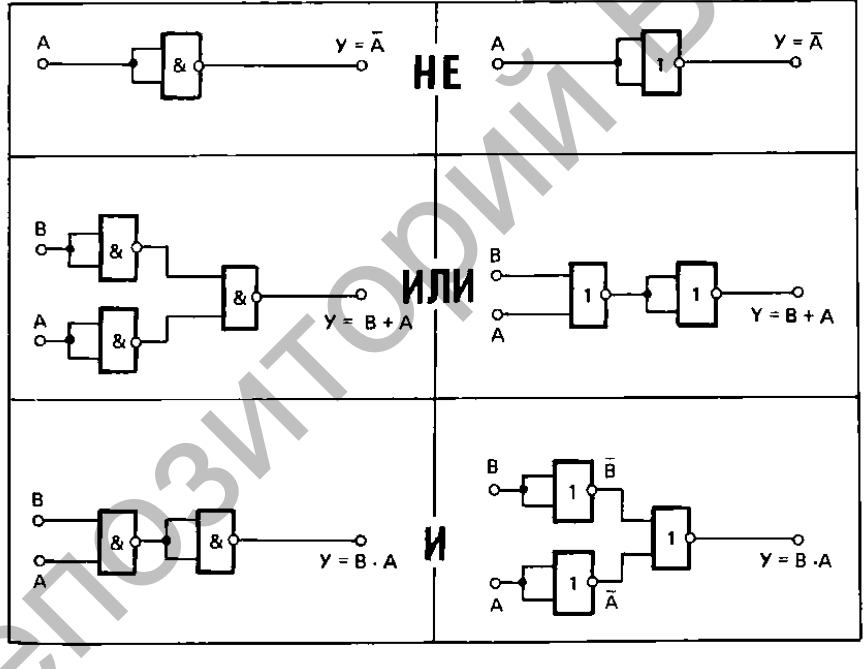


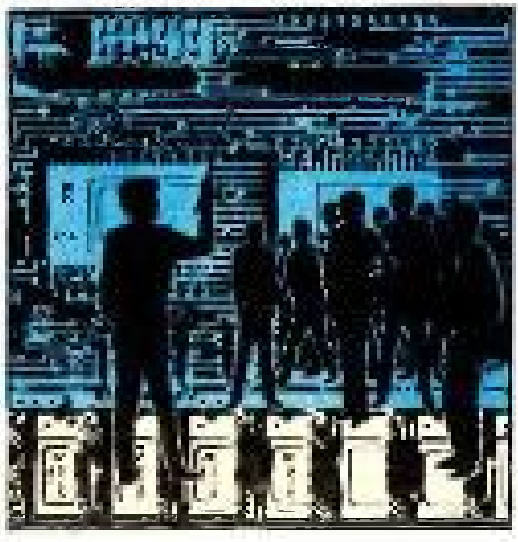
4.8. А как же с универсальной схемой и одной функцией?

Заявив в 4.4 о том, что любое устройство цифровой электроники можно построить из схем одного типа, мы почему-то забыли о такой возможности и синтезировали комбинационные схемы из схем И, ИЛИ, НЕ.

Все объясняется очень просто. Рассмотрен только один из возможных способов синтеза и минимизации. Он более прост и нагляден, чем другие, так как действие схем И, ИЛИ и НЕ легко пояснить при помощи простых аналогий (см. 4.2). Можно было бы рассмотреть способы синтеза и минимизации на основе схем одного типа, таких, как ИЛИ-НЕ, но мы поступим иначе. Покажем, что только из схем каждого из этих типов могут быть получены схемы И, ИЛИ и НЕ. Тем самым мы подтвердим, что, в принципе, из этих схем могут быть получены и любые комбинационные схемы. Как это может быть сделано — ясно из рисунка 4.7.

Рис. 4.7. С помощью законов инверсии (4.11) еще раз можно убедиться в универсальности двухвходовых схем И-НЕ и ИЛИ-НЕ.





Кадр 5

ЭЛЕКТРОННЫЕ ЦИФРОВЫЕ СХЕМЫ

Изобретенная в 1907 году трехэлектродная электронная лампа вплоть до середины нашего века была единственным устройством, позволяющим усиливать электрические колебания различной частоты.

В настоящее время известны десятки принципов, которые могут быть применены для этих целей, а также сотни методов их реализации. Но самым распространенным активным элементом электронной техники остается пока транзистор. В электронных цифровых схемах, как правило, используется только одно его свойство: способность включать или выключать большой электрический ток при помощи малого тока управления. Современные способы построения электронных устройств на транзисторах во многом напоминают тиражирование газет и фотоснимков.

Линейные и нелинейные элементы

Большинство электронных цифровых схем представляет собой *электрические цепи* из резисторов, диодов и транзисторов.

Резистор — элемент, проводящий электрический ток. Он имеет два вывода и вполне определенную величину электрического сопротивления R. Зависимость тока I, проходящего через резистор, от приложенного к нему напряжения U, подчиняется закону Ома

$$I = \frac{U}{R} \qquad (5.1)$$

График этой зависимости представляет прямую линию (рис. 5.1 *а*), и резистор относится к так называемым *линейным* элементам.

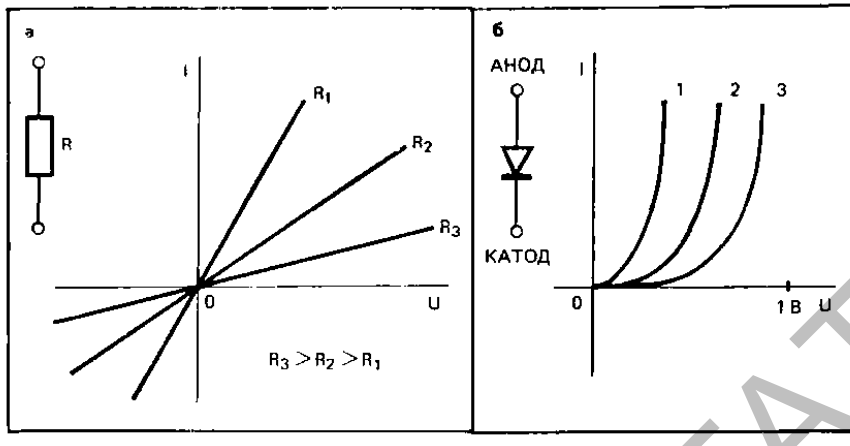


Рис. 5.1. Вольтамперные характеристики резисторов (*а*) и диодов (*б*): 1 — диод Шоттки, 2 — германиевый, 3 — кремниевый.

Простейший *нелинейный* элемент — *диод* также имеет два вывода. Он проводит ток, если к его аноду приложен положительный, а к катоду — отрицательный полюс источника электрического напряжения. В этом случае говорят, что диод смещен в прямом направлении.

Напряжение обратной полярности (плюс — к катоду, минус — к аноду) не вызывает тока через диод. Включенный таким образом диод считается смещенным в обратном направлении.

На рисунке 5.1, *б* представлены графически характеристики широко распространенных диодов: кремниевого, германиевого и диода, образованного переходом металл-полупроводник (диод Шоттки). Строго говоря, диод пропускает ток, хотя и очень малый, и в обратном направлении, но во многих случаях его можно не учитывать.

Транзистор — тоже нелинейный элемент.

5.2. Транзисторы

Транзистор — прибор с тремя внешними выводами: *эмиттером* (Э), *базой* (Б) и *коллектором* (К). Он изготавливается из полупроводникового материала, чаще всего из кремния. Наиболее быстродействующие транзисторы делаются из арсенида галлия, но пока они встречаются сравнительно редко.

Присоединим эмиттер транзистора к проводу, потенциал которого равен нулю. Такой провод называют общим, *«землей»* или *«корпусом»*. Если при этом коллектор должным образом подключен к источнику питания, то при некоторых условиях изменения тока базы приведут к точно таким же по характеру, но во много раз большим по величине, изменениям тока коллектора, т. е. ток

в цепи коллектора может быть во много раз увеличенной копией тока в цепи базы.

Переход база-эмиттер обладает ярко выраженными нелинейными свойствами. Его вольт-амперная характеристика подобна кривым на рисунке 5.1 *б*. Поэтому ток в цепи базы протекает только при положительных значениях переменного напряжения на базе. Отрицательные же значения напряжения независимо от их величины вообще не создадут никакого тока базы.

Для неискаженного преобразования напряжения на базе в ток переход база-эмиттер смещают в прямом направлении («приоткрывают») при помощи дополнительного источника постоянного напряжения — чаще всего части напряжения источника питания.

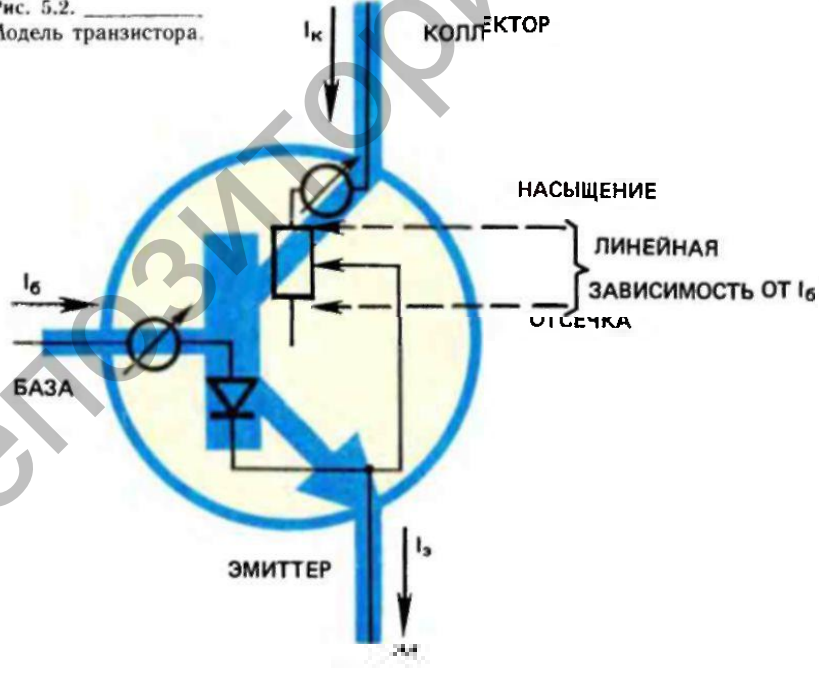
Цепь базы можно представить в виде диода, соединяющего базу с эмиттером, цепь коллектора — как переменный резистор между коллектором и эмиттером (рис. 5.2.). Таким образом, легко пояснить свойства транзистора в целом, но для этого нужно вообразить, что внутрь транзистора помещен... человек.

Человек смотрит на измеритель тока базы, умножает значение этого тока $I_б$ на постоянный положительный коэффициент β и регулятором величины переменного резистора устанавливает в цепи коллектора ток

$$I_к = \beta \cdot I_б$$

Заметим, что к такой модели прибегают даже в серьезных книгах для инженеров по радиоэлектронике, считая человека в транзисторе очень проворным, способным реагировать на изме-

Рис. 5.2.
Модель транзистора.



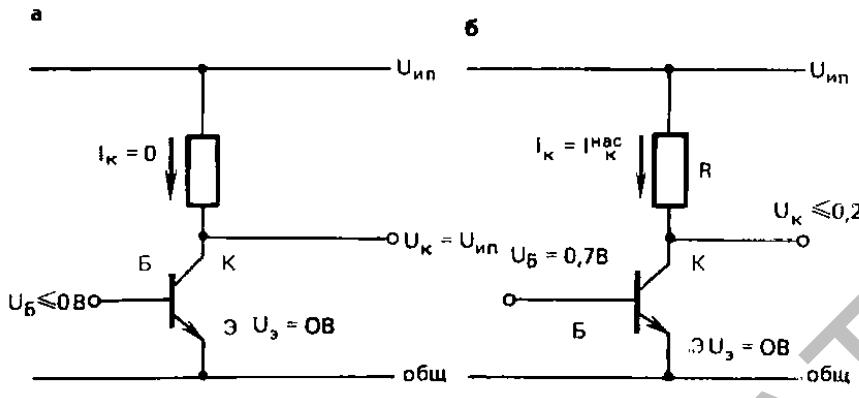


Рис. 5.3. Кремниевый транзистор в режиме отсечки *(а)* и насыщения *(б)*.

нения тока с частотой в десятки и даже сотни миллионов раз в секунду.

Коэффициент β вполне конкретен для каждого транзистора и равен обычно 100—250. Поэтому при возрастании тока базы до некоторого значения регулятор резистора окажется передвинутым в крайнее верхнее положение, и дальнейшее увеличение тока коллектора станет невозможным. Такой наибольший для данного транзистора ток коллектора называется *током насыщения*. Обозначим его $I_к^{max}$. Другая крайность — самое нижнее положение регулятора. Цепь коллектора при этом разрывается, и ток коллектора становится равным нулю. В таком случае говорят, что ток отсекается или что транзистор находится в *состоянии отсечки.*

Итак, коротко:

1) в схемах, в которых эмиттер присоединен к общему проводу («земле») или, что то же самое, в *схемах с общим эмиттером* ток базы усиливается в цепи коллектора до значения $I_к = \beta \cdot I_б$

2) нулевое и отрицательное напряжения на базе соответствуют режиму отсечки, при котором $I_к = 0$;

3) начиная с некоторого значения тока базы $I_б^{нас}$, ток коллектора достигает насыщения, и дальнейший рост тока базы не влияет на величину тока коллектора.

В большинстве бытовых приборов — радиоприемниках, телевизорах, магнитофонах, усилителях наиболее важным считается линейный режим работы транзистора, т. е. такой, при котором нет, во-первых, искажения входного сигнала и, во-вторых, не достигается как режим отсечки, так и режим насыщения.

Иное дело в цифровой электронике. Отсечка и насыщение связаны здесь, как правило, с сигналами 1 и 0. На рисунке 5.3 приведены значения напряжений на кремниевом транзисторе в случаях отсечки и насыщения.

5.3. О том же, но строже

Действие диодов и транзисторов основано на свойствах *полупроводниковых материалов.*

Типичный полупроводник — элемент четвертой группы таблицы Менделеева кремний (Si). Во внешней электронной оболочке атома кремния имеются четыре валентных электрона. Атомы кремния способны объединяться в очень прочную кристаллическую решетку при помощи химических ковалентных связей. В такой решетке каждый атом окружен четырьмя соседними атомами кремния и находится как бы в центре тетраэдра, образованного ими. Все четыре валентных электрона этого атома кремния и четыре электрона окружающих атомов («по одному от каждого ближайшего соседа») образуют единую электронную «орбиту». Эти электроны достаточно прочно связаны с атомами решетки, и для того чтобы их оторвать от нее, требуется большая энергия — порядка 1 эВ. Эта энергия существенно больше энергии теплового движения, поэтому при обычных температурах тепловое движение не может сообщить электрону скорость (иными словами энергия не может возрасти) до величины, достаточной для того, чтобы электрон оторвался от решетки и стал свободным. Поэтому в идеальном протяженном полупроводнике, в отличие от металлов, нет свободных носителей электрического заряда, и такой полупроводник не проводит электрический ток.

Однако на краю кристалла и в местах, где имеются дефекты кристаллической решетки, а также в тех местах, где вместо атомов кремния в решетку попали инородные атомы, стройная система ковалентных связей нарушается. С точки зрения электрических свойств полупроводника эти нарушения могут привести к преобладанию одного из двух эффектов.

В первом случае, который может быть обусловлен примесями третьей группы таблицы Менделеева, появляются «лишние» электроны. Энергия их связи с решеткой примерно в 100 раз меньше, чем у тех, которые участвуют в ковалентных связях, т. е. равна приблизительно 0,01 эВ. Поскольку энергия теплового движения при обычных температурах лежит в пределах 0,02—0,03 эВ, то, обладая такой энергией, эти лишние для химической связи электроны свободно перемещаются по кристаллу, т. е. становятся *свободными носителями заряда.*

Во втором случае, который может быть обусловлен вкраплением в решетку пятивалентных атомов, появляются «лишние связи». На них достаточно свободно переходят электроны с соседних атомов, а освободившуюся в результате такого перехода связь занимает электрон «другого атома» и т. д. Иными словами, по кристаллу может бродить «пустая связь», или *дырка*. Дырка эквивалентна потере электрона нейтральным атомом и обладает поэтому положительным зарядом. Энергия ее связи с решеткой также порядка 0,01 эВ.

Так в первом приближении может быть объяснено существова-

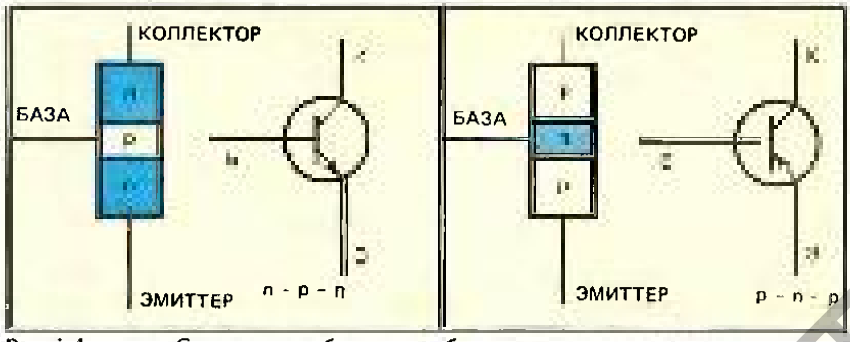


Рис. 5.4. Структура и обозначение биполярных транзисторов.

ние полупроводников двух типов — электронных, или *n-типа* и дырочных, или *p-типа* (от лат. *negativ* и *positiv* — отрицательный и положительный).

Если в одном кристалле создать граничащие между собой области проводимости *p*- и *n*-типа, то переход между ними (*p — n-переход*) обладает способностью проводить электрический ток только в одном направлении. Именно на основе *p—n-переходов* и изготавливаются полупроводниковые диоды, причем вывод от *p*-области называется по аналогии с вакуумными приборами *анодом,* а *n*-области — *катодом.*

Рассмотренный в 5.2 транзистор называется *биполярным,* т. е. использующим носители заряда обоих знаков. Его действие основано как на токе основных носителей (дырок в *p*-области или электронов в *n*-области), так и на токах неосновных носителей (дырок в *n*-области или электронов в *p*-области).

Биполярный транзистор представляет собой два близко расположенных *p—n*-перехода (рис. 5.4). Толщина центральной области (базы) составляет всего 1—2 микрометра и даже менее.

Проанализируем работу транзистора типа *n-p-n*. Основными носителями в эмиттерной и коллекторной областях являются электроны, в базовой — дырки.

При отключенном коллекторе переход база — эмиттер представляет собой обыкновенный диод и при положительном напряжении на базе через этот диод протекает электрический ток. Если же при этом подать на коллектор значительно большее положительное напряжение, то электроны эмиттера, попавшие в базу и продвинувшиеся на некоторое расстояние в сторону коллектора, окажутся в зоне действия сильного электрического поля коллектора. В реальных транзисторах из-за очень малой толщины базы большинство этих электронов будет собрано коллектором и лишь незначительная их часть (менее 1 %) нейтрализуется, объединяясь (рекомбинируя) с основными носителями базы — дырками.

Именно эта малая (~ 1 %) часть электронов (точнее — рекомбинирующих с электронами дырок) образует ток базы

$I_б$, а пропорциональная этому току, но значительно большая часть электронов, беспрепятственно преодолевающих область базы, образует ток коллектора $I_к$. Отношение приращений этих токов и есть коэффициент

$$\beta = \frac{\Delta I_к}{\Delta I_б}. \qquad (5.2)$$

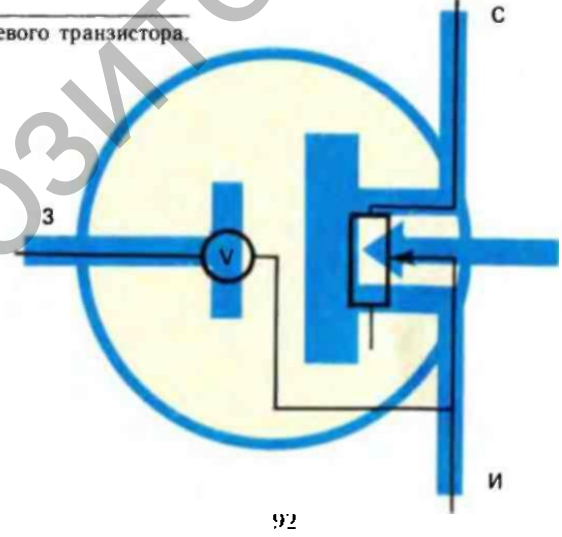
5.4. Полевые транзисторы

Полевые транзисторы иногда называют МДП или МОП по первым буквам слов *металл-диэлектрик-полупроводник, металл-оксид-полупроводник,* отражающих технологию их изготовления. МОП- и МДП- транзисторы лишь отдельные представители транзисторов, в которых током управляет электрическое поле.

Полевой транзистор имеет три основных вывода: *исток* (И), *затвор* (З) и *сток* (С). Управляющим электродом служит затвор. У МДП-транзисторов есть еще один внешний вывод — *подложка* (П), его обычно соединяют с истоком и не рассматривают отдельно. В простейшем случае исток подключается к общему проводу. К стоку подводится напряжение от источника питания. Ток между истоком и стоком зависит от напряжения на затворе, точнее от напряжения затвор-подложка, так как именно это напряжение обусловливает величину электрического поля, в котором находится зона перехода от истока к стоку. В отличие от биполярных транзисторов управление происходит, практически, без затраты мощности управляющего сигнала.

В модели полевого транзистора с человеком-регулировщиком (рис. 5.5) человек должен следить не за током, а за напряжением на затворе и в зависимости от величины этого напряжения

Рис. 5.5.
Модель полевого транзистора.



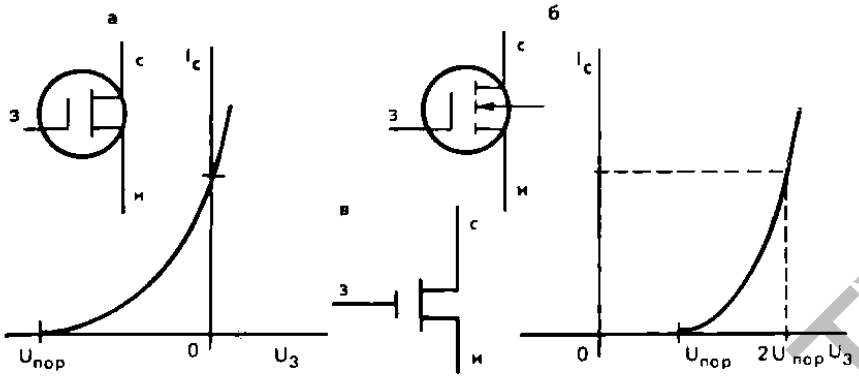


Рис. 5.6. Обозначение и передаточные характеристики полевых транзисторов: *а* обедненного (нормально открытого), *б* обогащенного (нормально закрытого). В первом случае максимально допустимый ток наблюдается при $U_з = 0$, во втором — при $U_з = 2U_{пор}$. Нередко для обозначения всех видов полевых транзисторов применяют общий символ *(в)*.

устанавливать ток в цепи исток-сток при помощи переменного резистора.

Различают два основных типа полевых транзисторов (рис. 5.6). В первом из них зависимость тока стока I_c от напряжения на затворе $U_з$ имеет вид кривой (рис. 5.6, *а*). Такой транзистор называют *нормально открытым*, так как при напряжении затвора, равном нулю, он проводит ток. Второй тип полевого транзистора — *нормально закрытый*. При нулевом потенциале на затворе ток стока транзистора равен нулю (рис. 5.6, *б*).

В цифровых электронных устройствах используют преимущественно нормально закрытые полевые транзисторы.

Имеется определенная аналогия между биполярным *n-p-n*-транзистором и полевым нормально закрытым транзистором с каналом *n*-типа: оба они не проводят ток при нулевом (и отрицательном) смещении на управляющем электроде, оба открываются при положительном смещении на этом электроде, оба достигают тока насыщения.

Подобного рода аналогию можно усмотреть и в характеристиках биполярного *p-n-p*- транзистора и нормально закрытого полевого транзистора с каналом *p*-типа.

5.5. Базовые логические элементы

Базовый набор должен содержать элементы, из которых можно построить любое цифровое электронное устройство. Поэтому, в принципе, в таком наборе могут быть только двухвходовые схемы И-НЕ (или только ИЛИ-НЕ) при условии, что выход каждой из них можно подключать к любому количеству входов

других таких же схем. В действительности число таких подключений ограничено, и уже только по этой причине в базовом наборе должны содержаться схемы, позволяющие расширить возможности отдельного выхода. Они так и называются — *расширители выхода,* а с точки зрения логики представляют собой повторители сигналов. Обычно же в базовом наборе наряду с элементами И-НЕ (или ИЛИ-НЕ) содержатся и некоторые другие схемы. Всех их объединяет идентичность входных и выходных сигналов.

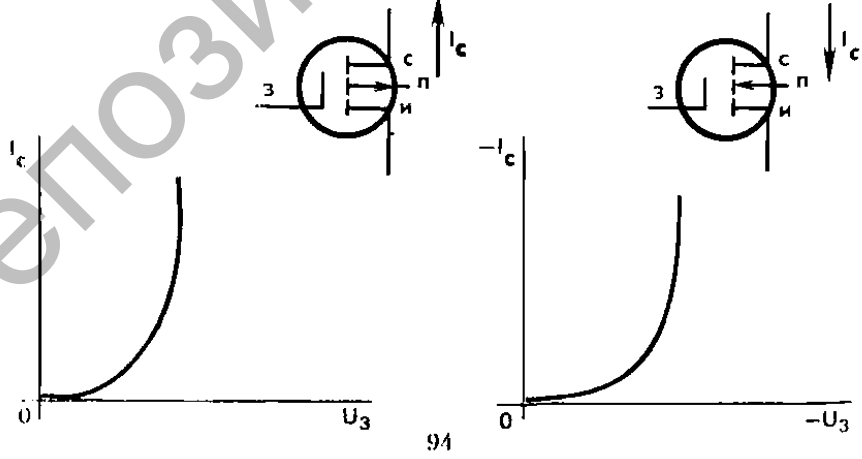
Выпускаемые промышленностью цифровые схемы имеют два типа выводов: к первым подключаются источники питания, ко вторым — логические входы и выходы.

В элементах *транзисторно-транзисторной логики* (ТТЛ) за 0 принято 0 В, за 1...5 В, напряжение источника питания также 5 В. Допускается и некоторое отклонение сигналов 0 и 1 от своих точных значений: до (0...0,4) В для 0 и (2,5...5) для 1. В свое время эти элементы получили столь широкое распространение, что для обмена сигналами между любыми цифровыми устройствами (ЭВМ, микропроцессорами и т. п.) приняты уровни логических сигналов ТТЛ.

Наименьшим потреблением энергии отличается *комплементарная МОП-логика* или сокращенно — КМОП-логика. В ней за логический 0 принято напряжение 0 В, за 1 — напряжение источника.

Комплементарной (т. е. взаимодополняющей) считают пару полевых транзисторов, имеющих одинаковые характеристики, но противоположную полярность питающих напряжений, сигналов управления и токов исток-сток (рис. 5.7). Если такую пару соединить так, как показано на рисунке 5.8 (обратите внимание, верхний транзистор «перевернут»), то:

Рис. 5.7. Характеристики комплементарной пары полевых транзисторов идентичны и различаются лишь полярностью действующих напряжений и направлениями тока стока. На схемах их различают по стрелке четвертого вывода (подложки — П).



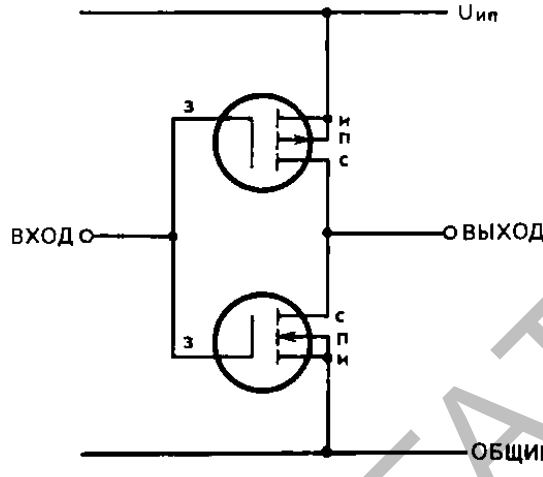


Рис. 5.8. КМОП инвертор (схема НЕ): 0 на входе закрывает верхний и открывает нижний транзистор, 1($U_{вх}$ = = $U_{ип}$) — наоборот.

1) при напряжении на объединенном затворе (входе), равном 0 В, верхний транзистор будет открыт, а нижний закрыт;

2) при напряжении на этом же входе, равном напряжению источника питания $U_{ип}$, наоборот, верхний открыт, а нижний — закрыт.

У открытого транзистора сопротивление исток-сток мало (в модели на рис. 5.5 регулятор переменного резистора находится в крайнем верхнем положении), у закрытого — велико (нижнее положение регулятора, т. е. разрыв цепи исток-сток), поэтому напряжение на выходе схемы на рис. 5.8 в первом случае будет $U_{ип}$, во втором 0 В. Схема представляет собой элемент НЕ.

Действие элементов И-НЕ и ИЛИ-НЕ КМОП-логики основано на следующем: во-первых, если группа транзисторов соединена параллельно (объединены истоки и стоки), то достаточно открыть хотя бы один транзистор и общее сопротивление исток-сток станет малым, т. е. цепь исток-сток окажется замкнутой; во-вторых, если группа транзисторов соединена последовательно (сток первого к истоку второго, сток второго — к истоку третьего и т. д.), то эта последовательная цепь разомкнута при условии, что хотя бы один транзистор закрыт. Помня об этом, нетрудно уяснить принцип действия элементов И-НЕ и ИЛИ-НЕ, показанных на рисунке 5.9. Заметим, что параллельно соединены транзисторы с индексом А, последовательно — с индексом В.

Системы логических элементов постоянно совершенствуются. Популярные в недавнем прошлом *резисторно-транзисторная логика* (РТЛ) и *диодно-транзисторная логика* (ДТЛ) вообще не применяются. Быстродействие схем ТТЛ удалось значительно повысить, благодаря введению в них диодов Шоттки. Задержка сигналов в элементах ТТЛ-Шоттки доведена до 2 нс. Однако самой быстродействующей остается пока *эмиттерно-связанная логика* (ЭСЛ).

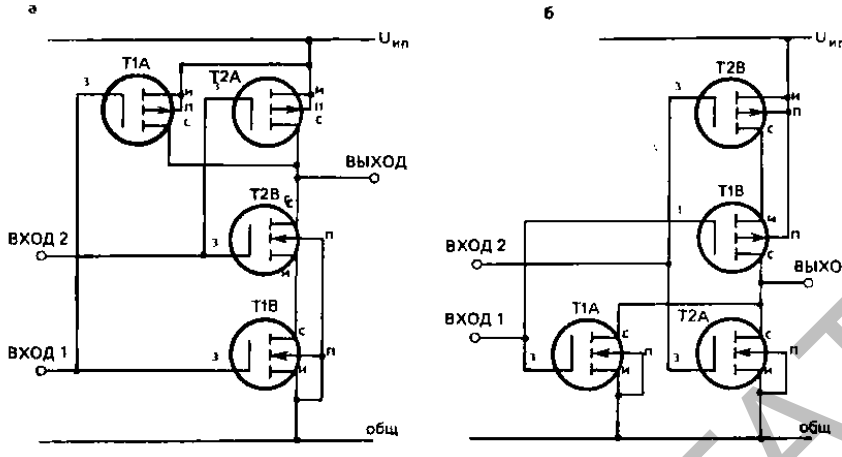


Рис. 5.9. Схемы ИЛИ-НЕ *(а)* и И-НЕ *(б)* типа КМОП.

Элементы так называемой *интегральной инжекционной логики* ($И^2Л$) — наиболее экономичные из устройств на биполярных транзисторах. Однако, самое малое потребление энергии от источников питания обеспечивает КМОП-логика.

5.6. Интегральные схемы

Первый транзистор был изготовлен в 1948 году. Через 20 лет электронная промышленность мира выпускала в год сотни миллионов транзисторов различных типов, и сфера их применения все расширялась. Во многих случаях, например для изготовления цифровых логических схем, нет необходимости делать транзисторы в виде отдельного прибора с внешними выводами, или, что то же самое, выпускать транзисторы в дискретном исполнении. Более надежными, экономичными и существенно более компактными получаются устройства, в которых на одной полупроводниковой пластинке формируется много транзисторов, диодов и резисторов, а перемычки между ними получают путем металлизации, например методом вакуумного напыления металлических проводников через маску нужной конфигурации.

Устройства такого типа, заключенные в герметичный корпус с внешними выводами, называются *интегральными схемами* (ИС).

Современные ИС — результат освоения производством весьма совершенных технологических процессов, базирующихся на достижениях фундаментальных наук, прежде всего физики.

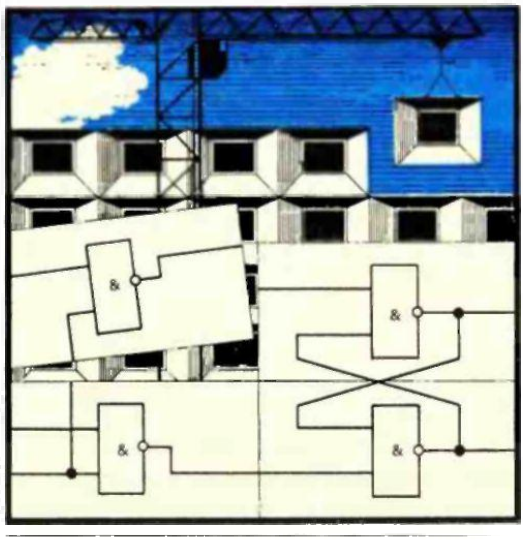
Материалом для большинства ИС служит самый распространенный на земле элемент — кремний, подвергнутый тщательной очистке. Чтобы представить степень этой очистки, заметим, что промышленностью освоено производство монокристаллов кремния, в которых на число атомов, равное населению Земли ($5 \cdot 10^9$), приходится всего 3—4 атома примеси.

Монокристалл кремния разрезают на очень тонкие пластины, которые шлифуются и полируются до зеркального блеска. На 1 мм^2 такой пластины получают более тысячи транзисторных структур. Некоторые большие ИС (БИС) на кристаллах размерами 5×6 мм^2 содержат более 100 000 активных элементов и представляют собой целые универсальные ЭВМ.

Применяемую сейчас технологию производства интегральных схем можно назвать «одноэтажной», так как активные элементы образуют на полупроводниковой пластине только один слой. Она достигла высокой степени совершенства, и расчеты показывают, что плотность расположения компонентов, которую предполагают получить в ближайшие годы, сравняется с теоретически достижимым пределом.

Интенсивно ведется поиск принципиально новых способов построения интегральных схем, позволяющих создавать на одной подложке «многоэтажные» структуры и даже объемные.

Перспективы дальнейшего совершенствования интегральных схем связывают с целым рядом интересных физических явлений, новых материалов и технологий. Решительное повышение быстродействия сулит переход от электронных устройств к оптическим или хотя бы электронно-оптическим. Другое направление — работа при низких и очень низких (криогенных) температурах. В этом случае наряду с повышением быстродействия резко снижается потребляемая мощность, еще более уменьшаются габариты интегральных схем. Большие исследовательские группы ищут пути синтеза «интегральных схем» на молекулярном и субмолекулярном уровнях. Какое направление получит право на жизнь — покажет будущее.



Глава 6

6.1. Что дают обратные связи? 6.2. Синхронизируемые триггеры. 6.3. «Делай, как я!» или о разных триггерах. 6.4. «Прыгай — держись!» 6.5. Регистры, счетчики. 6.6. Шифраторы, дешифраторы, мультиплексоры. 6.7. Как считает ЭВМ? 6.8. Электронная память, или СОЗУ, ОЗУ, ПЗУ, ППЗУ и т. п. 6.9. Магистраль. 6.10. Окно в аналоговый мир, или ЦАП и АЦП. 6.11. Еще раз о кодах.

ЭЛЕКТРОННЫЕ УСТРОЙСТВА

Гораздо быстрее, чем из кирпичей, возводят здания из крупных панелей, блоков. Благодаря искусству архитекторов даже домам из таких типовых строительных деталей свойственна определенная индивидуальность.

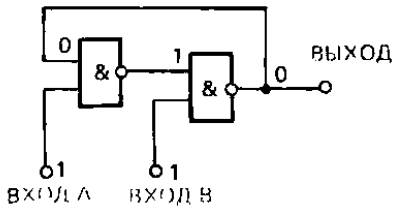
Что-то похожее наблюдается и в вычислительной технике. В принципе, можно построить электронные устройства и непосредственно из «кирпичей» типа элементов И-НЕ, но значительно удобнее использовать «детали» более крупные. Каждая такая деталь в целом выполняет вполне определенные функции, что и закрепило за ними название электронные функциональные элементы и устройства. Системы из таких автономных функциональных блоков проще конструировать, налаживать, обслуживать, ремонтировать.

6.1. Что дают обратные связи?

Обратная связь, или воздействие выходного сигнала на вход управления, играет важную роль в радиотехнике, электронике, системах автоматического управления и регулирования (см. 1.4). Устройства с обратной связью лежат в основе построения высококачественных усилителей, измерительных приборов, автоматов, радиолокаторов и т. п.

Так как сигналы, вырабатываемые цифровыми схемами, однотипны с сигналами управления этими схемами, то цепи обратной связи в цифровой электронике предельно просты. Они представляют собой соединения выходов со входами.

В некоторых случаях цифровая обратная связь позволяет как бы «удержать» сигнал на выходе после окончания воздействия



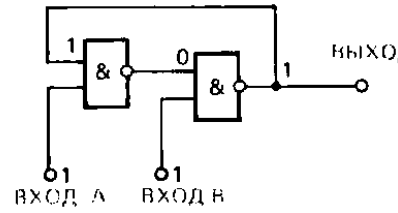
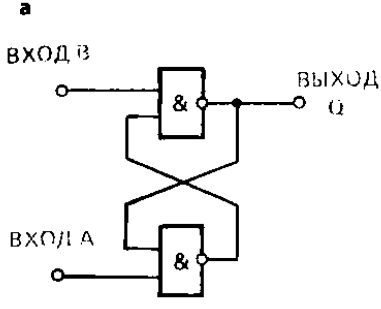


Рис. 6.1 Два последовательно соединенных элемента И-НЕ образуют триггер. В режиме хранения на оба входа *A* и *B* поданы логические 1, но в первом случае триггер хранит 0, во втором — 1. Оба состояния устойчивы.



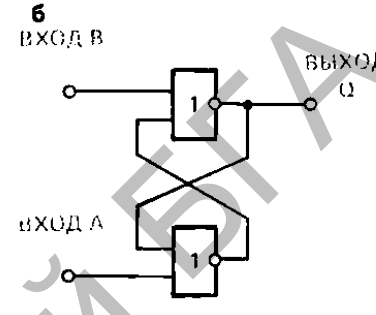


Рис. 6.2 Собственно триггер на схемах И-НЕ *(а)* и на схемах ИЛИ-НЕ *(б)*. В первом случае для хранения информации на входы *A* и *B* подают логические 1, во втором — логические 0.

входного сигнала, т. е. придает схеме способность сохранять состояние 0 или 1. Схема, способная хранить («запоминать») 0 или 1, называется *триггером*. Триггер — простейший цифровой автомат с памятью (см. 2.3, 2.4).

На рисунке 6.1 показано, как построить триггер из двух схем И-НЕ. Обычно же триггеры изображают так, как показано на рисунке 6.2, и называют собственно триггерами. Собственно триггер — обязательная составная часть любых более сложных триггеров.

Далее будет использоваться только триггер на элементах И-НЕ (рис. 6.2, *а*). Его свойства поясняет таблица 6.1, в которой исходное состояние триггера обозначено Q_t, а новое — Q_{t+1}.

Таблица 6.1

B	*A*	Q_{t+1}
0	0	неопред.
0	1	1
1	0	0
1	1	Q_t

B \ *A*	0	1
0	?	1
1	0	Q_t

Предполагается, что после каждого воздействия набора входных сигналов B и A триггер переводится в режим хранения, т. е. на входы B и A одновременно подаются логические 1.

6.2. Синхронизируемые триггеры

Различают два типа входов триггеров — *информационные* и *синхронизирующие*. Сигналы на информационных входах и исходное состояние Q_t определяют, каким будет новое состояние триггера. Сигналы на синхронизирующих входах определяют время переключения.

Простейший синхронизируемый триггер показан на рисунке 6.3. Он называется *RS-триггером* (от англ. *set*, *reset* — установить, переустановить). R- и S-входы — информационные. C-вход — синхронизирующий (от англ. *clock* — часы, засекать время). На него постоянно подан 0, поэтому на выходах 1-й и 2-й схем И-НЕ — логические 1, и собственно триггер находится в режиме хранения информации. Если же на вход C на короткое время подать логическую 1 (C-импульс), то собственно триггер установится в состояние, зависящее от сигналов на входах R и S. Все возможные варианты переходов отражены в таблице 6.2.

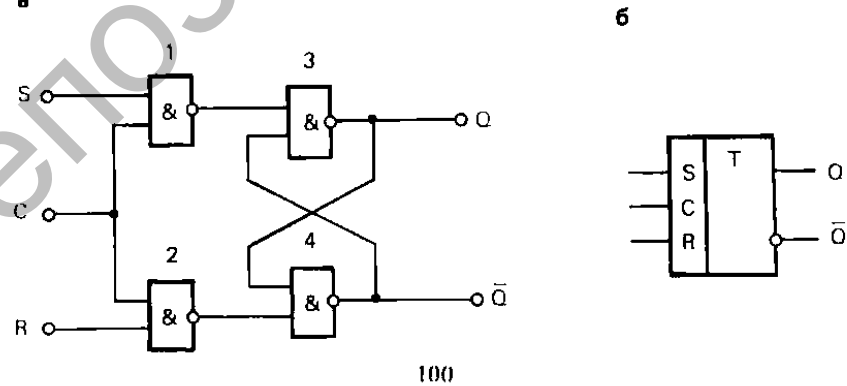
Таблица 6.2

S	R	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	неопред.

S \ R	0	1
0	Q_t	0
1	1	?

Состояние триггера при наличии логической единицы на обоих входах не определено, потому что бессмысленно требовать установки и сброса триггера одновременно. Состояние реального триггера при таких входных сигналах зависит от его устройства.

Рис. 6.3. Синхронизируемый RS-триггер: *а* — структура, *б* — обозначение.



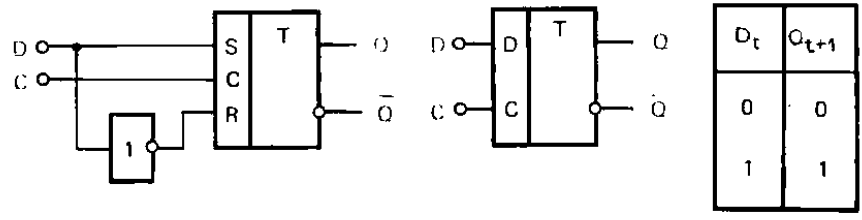


Рис. 6.4. D-триггер: *а* — структура, *б* — обозначение, *в* — таблица состояний.

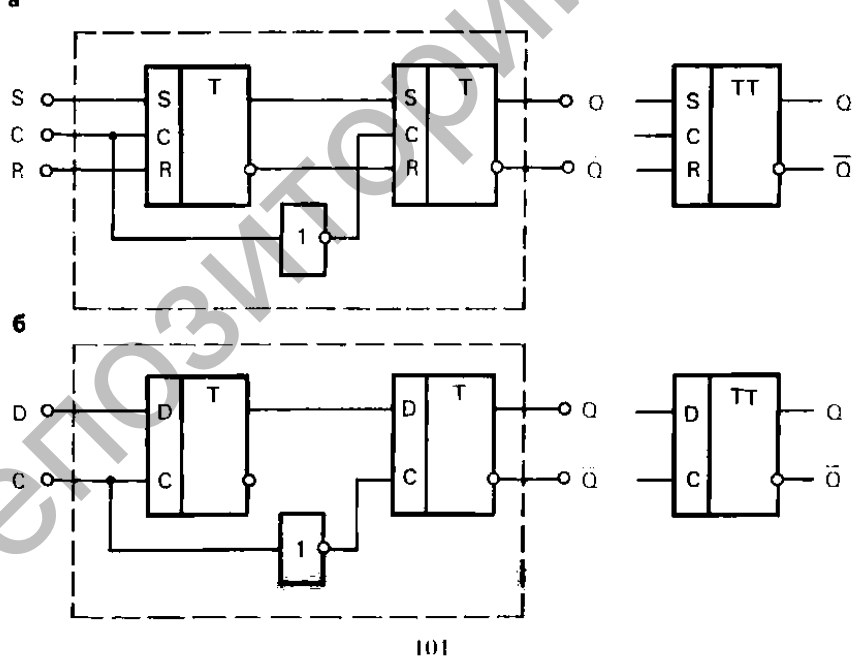
Триггер задержки или *D-триггер* (от англ. *delay* — задержка) имеет только один информационный вход. Его схема приведена на рисунке 6.4. Легко убедиться, что после *C*-импульса он повторяет сигнал на входе *D*.

6.3. «Делай, как я!» или о разных триггерах

Обратная связь помогла превратить комбинационную схему в триггер. А не способна ли она расширить возможности самого триггера?

Попытка ввести обратную связь в рассмотренные синхрони-

Рис. 6.5. Структура и обозначение триггеров с двухступенчатым запоминанием RS-типа *(а)* и D-типа *(б)*.



зируемые триггеры наталкивается на определенные трудности, так как сигнал обратной связи может измениться до окончания импульса синхронизации и привести к повторному, т. е. незапланированному, переключению триггера.

Чтобы исключить подобного рода нежелательные явления, применяют триггеры с двухступенчатым запоминанием. Их называют также *MS-триггерами* по первым буквам английских слов *master-slave* — хозяин-раб, подчеркивая тем самым, что триггер состоит из двух частей, одна из которых (*master*) как бы заставляет другую (*slave*) повторить свои действия. Более четко суть происходящих процессов отражают термины ведущий-ведомый: ведомый поступает так, как ведущий.

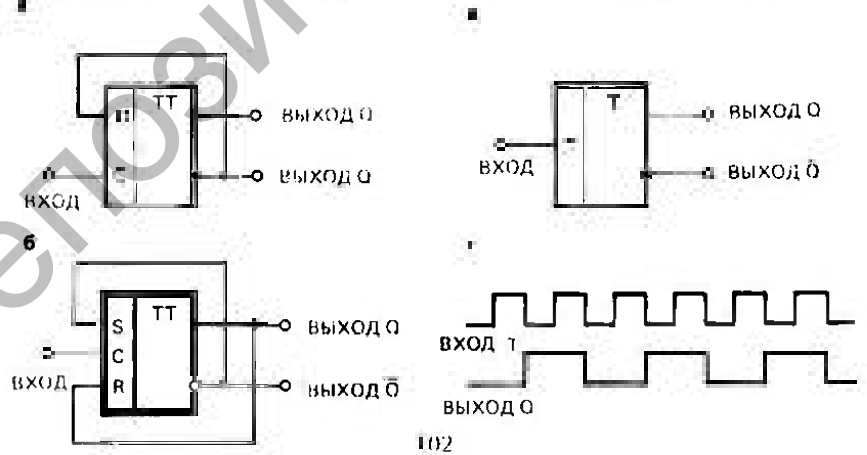
На рисунке 6.5 показана структура *RS*- и *D*-триггеров с двухступенчатым запоминанием. Каждый из них содержит по два обычных синхронизируемых триггера и описывается той же таблицей состояний, что и обыкновенный триггер эквивалентного типа. В отличие от обычных *RS*- и *D*-триггеров сигнал переключения появляется на выходе триггера с двухступенчатым запоминанием после окончания импульса синхронизации: в начале *C*-импульса переключается ведущий триггер, в момент окончания — ведомый.

6.4. «Прыгай-держись!»

Аббревиатура *JK*, входящая в название наиболее универсального триггера, обязана своим происхождением английским словам *Jump-Keep* — прыгай-держись.

Изучение *JK-триггера* начнем с анализа некоторых способов включения уже известных нам *RS*- и *D*-триггеров с двухступенчатым запоминанием. В этих триггерах, как и во всех остальных, наряду с основным обычно предусматривается и инверсный выход $\bar{Q}$. Он не несет никакой дополнительной информации, но

Рис. 6.6. Организация счетного входа у триггеров с двухступенчатым запоминанием.



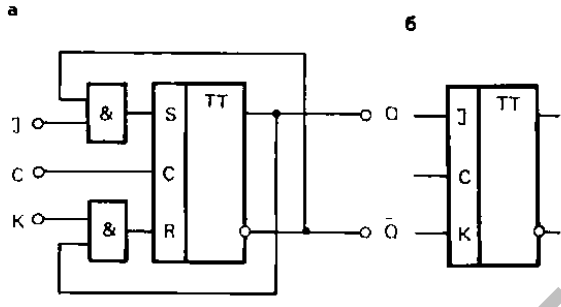


Рис. 6.7. Структура JK-триггера *(а)* с двухступенчатым запоминанием и его обозначение *(б)*.

в ряде случаев удобен как источник сигнала, противоположного (инверсного) тому, который наблюдается на основном выходе.

В схеме на рисунке 6.6, *а* инверсный выход *D*-триггера с двухступенчатым запоминанием соединен с его же информационным входом, и поэтому после *C*-импульса *D*-триггер переходит из состояния Q в противоположное состояние $\bar{Q}$. После второго *C*-импульса триггер возвращается в исходное состояние. Иными словами, на выходе такого триггера в ответ на два импульса на входе появляется один импульс на выходе (рис. 6.6, *г*).

Такой режим работы триггера как и образовавшийся при этом *T*-вход называют *счетным*. На рисунке 6.6, *б* показано, как можно получить счетный вход в *RS*-триггере, на рисунке 6.6, *в* дано обозначение таких триггеров.

Структура *JK*-триггера (рис. 6.7) напоминает двухступенчатый *RS*-триггер с обратными связями для получения счетного входа. Однако обратные связи заведены не непосредственно на *R*- и *S*-входы, а через схемы И. На другие входы этих схем И подаются соответственно информационные сигналы *J* и *K*. Если $J=1$ и $K=1$, то триггер полностью эквивалентен схеме, показанной на рисунке 6.6,б, и работает как счетный. Свойства *JK*-триггера пояснены таблицей 6.3.

Таблица 6.3

J	K	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	$\bar{Q}_t$

J \ K	0	1
0	Q_t	0
1	1	$\bar{Q}_t$

6.5. Регистры, счетчики

Регистр — это цепочка триггеров для запоминания одного двоичного числа. Общее количество триггеров равно наибольшей разрядности хранимого числа. Каждому триггеру может быть поставлен в соответствие весовой коэффициент или просто вес (см. 3.1). Если все триггеры регистра находятся в состоянии 0,

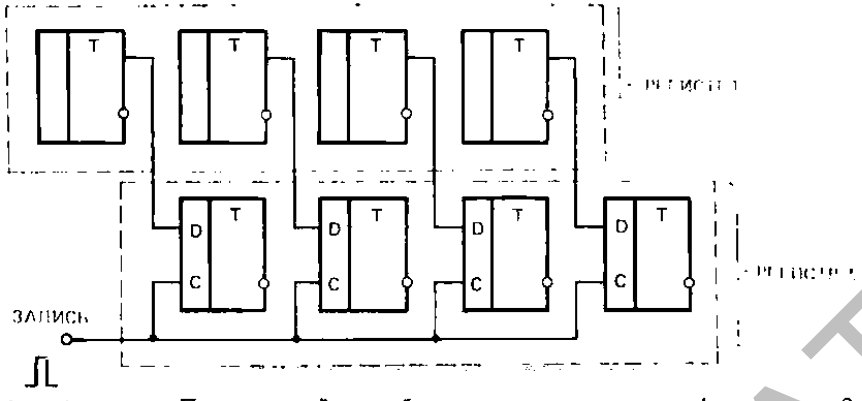


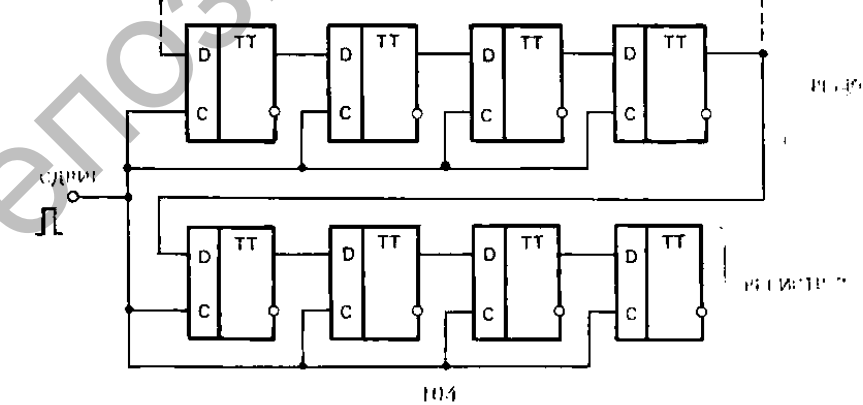
Рис. 6.8. Параллельный способ передачи кода регистра 1 в регистр 2.

а данный триггер — в состоянии 1, то хранимое в регистре число равно весу этого триггера. В чисто двоичном n-разрядном регистре веса триггеров равны 2^0, 2^1, ..., 2^{n-1}, т. е. 1, 2, 4, 8, ...

Код числа можно передать из одного регистра в другой параллельным или последовательным способом. Первый способ поясняется рисунком 6.8. Кратковременное появление логической 1 на объединенном C-входе второго регистра заставляет каждый триггер этого регистра устанавливаться в состояние, соответствующее сигналу на D-входе, т. е. в состояние связанного с ним триггера первого регистра.

При последовательном способе передачи (рис. 6.9) на объединенный C-вход первого и второго регистров подают n импульсов. Каждый C-импульс устанавливает данный триггер в состояние соседа слева, поэтому после n импульсов (4 для случая, представленного на рис. 6.9) код первого регистра будет передан во

Рис. 6.9. Последовательный способ передачи кода из регистра 1 в регистр 2.



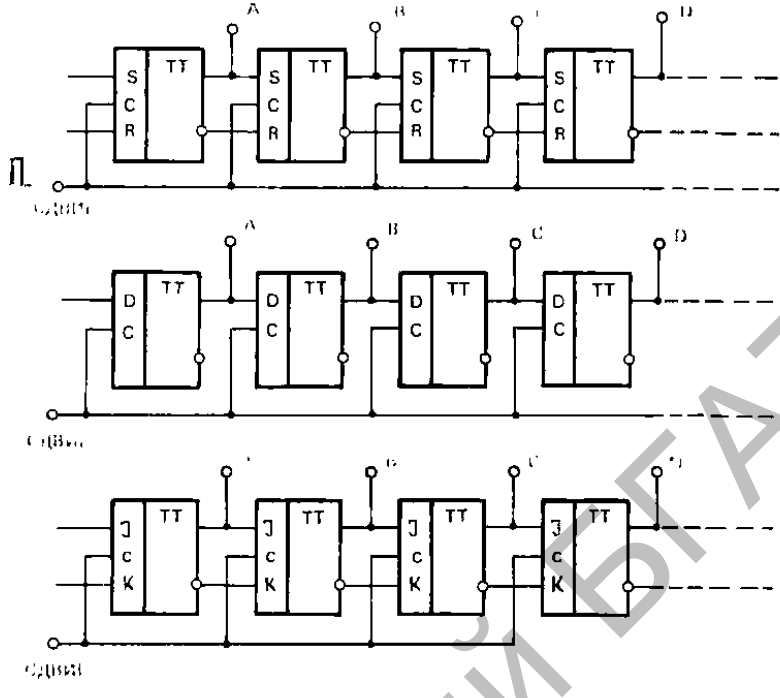


Рис. 6.10 Сдвигающие регистры на RS-, D- и JK-триггерах с двухступенчатым запоминанием.

второй. Когда при этом необходимо сохранить содержимое в первом регистре, то делается соединение, показанное на рисунке 6.9 пунктиром.

В *сдвигающем регистре* (рис. 6.10) при каждом импульсе управления весь код смещается по отношению к цепочке триггеров на одну позицию. Если веса триггеров фиксированы, то сдвиг в одну сторону эквивалентен умножению числа на 2, в другую — делению на 2.

Одна из разновидностей регистров — пересчетные схемы или просто *счетчики*. Каждый импульс на входе суммирующего счетчика (рис. 6.11, *а*) увеличивает код хранимого в нем числа на 1. В вычитающем счетчике импульсы вычитаются из содержимого счетчика (рис. 6.11, *б*). *Реверсивный счетчик* (рис. 6.11, *в*) объединяет в себе свойства и того и другого, имея вход суммирования «+» и вход вычитания «—» (сравни рис. 3.10. и рис. 3.12).

Счетчик из n триггеров имеет 2^n устойчивых состояний и может хранить числа от 0 до $2^n - 1$. Величину $m = 2^n$ называют *коэффициентом пересчета*. Различными способами (например, введением линий обратной связи) некоторые из устойчивых состояний можно запретить. Тогда итоговый коэффициент пересчета станет меньше, чем 2^n. Счетчик, в котором реализуется десять

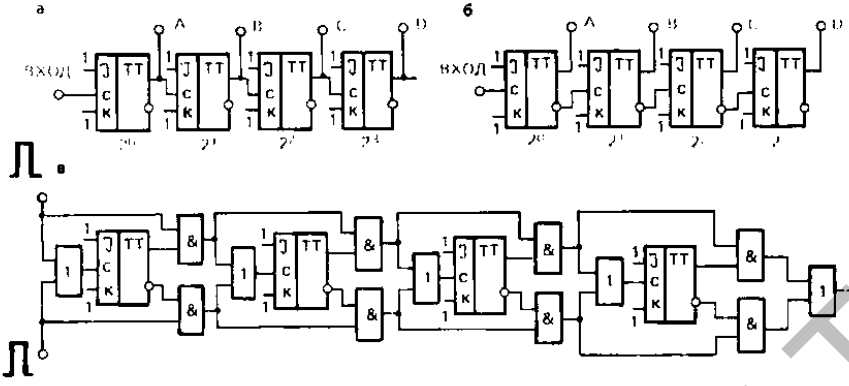


Рис. 6.11. В простейших счетчиках при поступлении на вход очередного импульса все подлежащие переключению триггеры изменяют свое состояние не одновременно, а последовательно друг за другом, поэтому такие пересчетные схемы называют асинхронными.

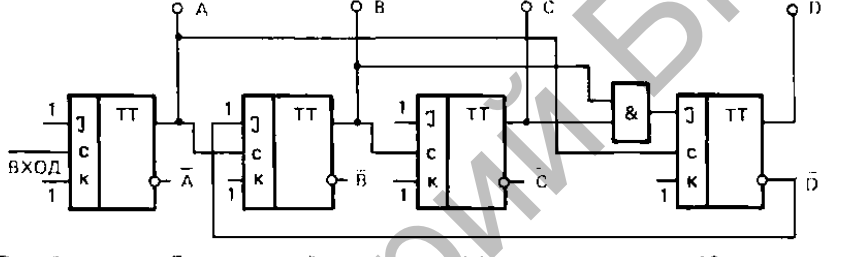


Рис. 6.12. Суммирующий счетчик с коэффициентом пересчета 10 («декада»).

устойчивых состояний, называют десятичным или *декадой* (рис. 6.12). Несколько включенных друг за другом декад образуют многоразрядный счетчик, работающий в привычной десятичной системе счисления.

Заметим, что сами по себе электронные счетчики широко применяются в физических исследованиях,особенно в ядерной физике, где скорость счета событий нередко исчисляется сотнями миллионов и даже миллиардами в секунду.

6.6. Шифраторы, дешифраторы, мультиплексоры

Установить, в каком из 2^n возможных состояний находится n-разрядный регистр, позволяет *дешифратор* (от фр. *dechiffrer* — расшифровывать). Он имеет n входов и 2^n выходов. При любой комбинации сигналов на входах сигнал 1 наблюдается только на одном из выходов, т. е. каждое из 2^n возможных состояний регистра обусловливает появление 1 на «своем» выходе. Выходы принято обозначать таким образом, что индекс при букве y является десятичным аналогом распознаваемого двоичного кода.

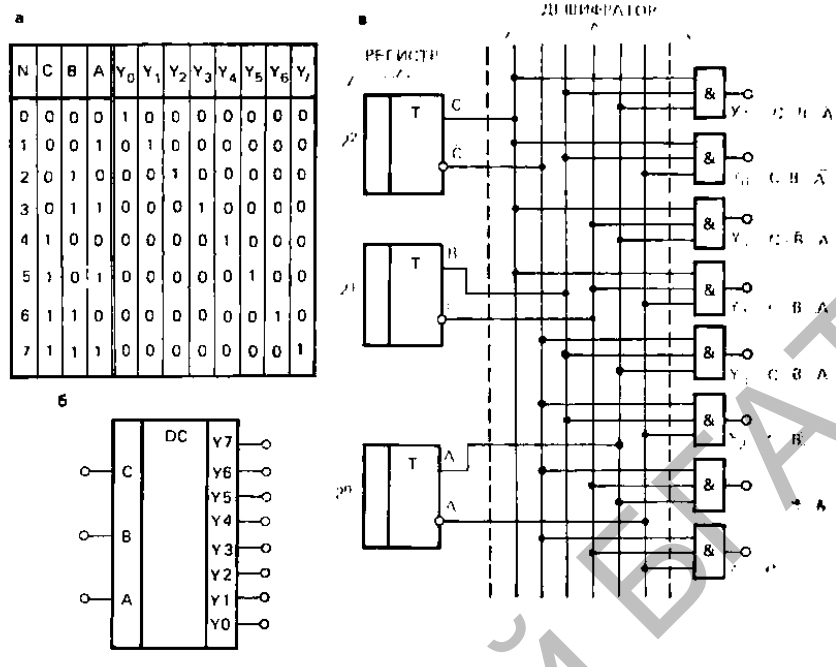


Рис. 6.13. Трехразрядный двоичный дешифратор: *а* — таблица состояний, *б* — обозначение, *в* — структура и принцип подключения к регистру.

В простейшем случае дешифратор представляет собой набор из 2^n n-входовых схем И. На входы каждой из них поступают прямые или инверсные сигналы с выходов всех триггеров регистра (рис. 6.13).

Эта схема может рассматриваться как преобразователь двоичного натурального кода в код «1 из n» (см. 3.1). Обратную функцию производит *шифратор*. Логическая 1 на любой из вертикальных шин (рис. 6.14) преобразуется в параллельный двоичный код. В этом легко убедиться, вспомнив принцип действия схемы ИЛИ. Аналогично осуществляется преобразование и в коды других типов, а комбинация устройств дешифратор-шифратор позволяет преобразовывать код одного типа в другой код. Рисунок 6.15 поясняет, как преобразуется двоичный код в код управления сегментным индикатором.

Слово *мультиплексор* заимствовано из английского и обозначает молоточек для выстукивания пациентов. Звук от ударов по различным точкам тела воспринимается врачом на слух. Таким образом, различные источники информации (точки тела) передаются для анализа через один и тот же канал (ухо врача).

По аналогии использование одних и тех же шин для передачи информации от различных источников называют мультиплексиро-

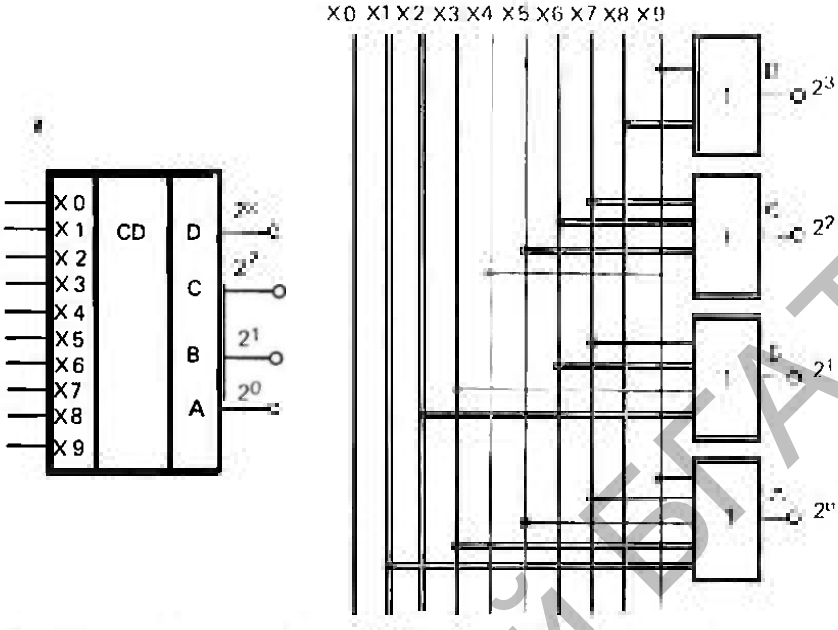
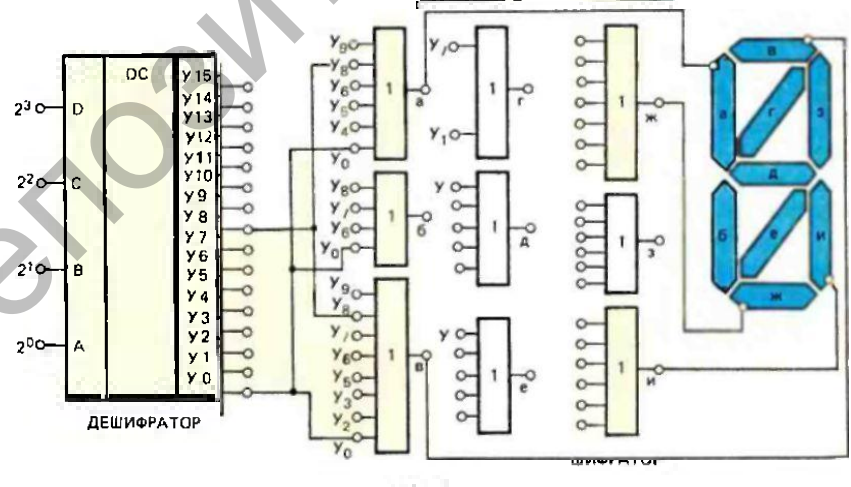


Рис. 6.14. Обозначение *(а)* и структура *(б)* двоичного шифратора.

ванием, а устройства сведения информации в одну шину — мультиплексорами. Обратная операция — разделение информации по адресам назначения производится при помощи *демульти-*

Рис. 6.15. Принцип преобразования параллельного двоичного кода *DCBA* в код управления сегментным индикатором абвгдежзи (часть соединительных проводов на рисунке не показана).



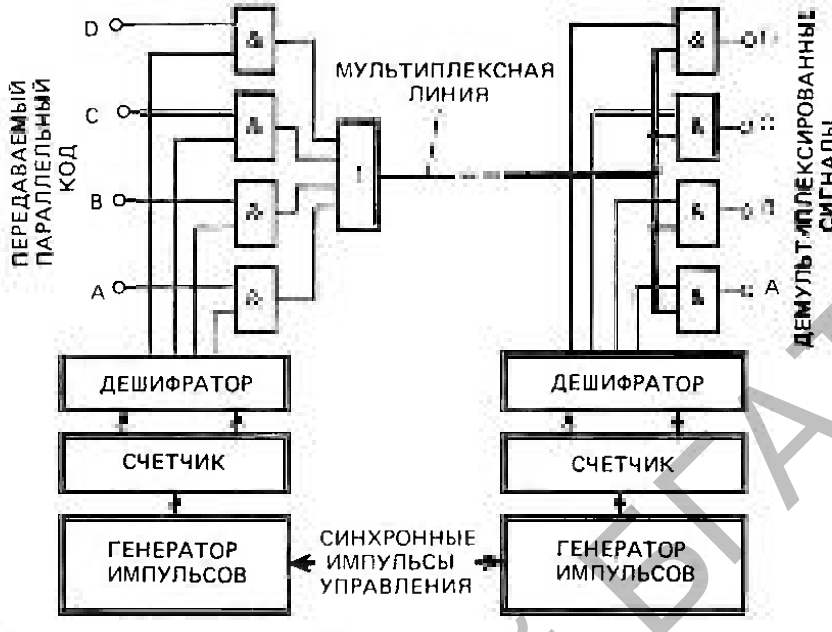


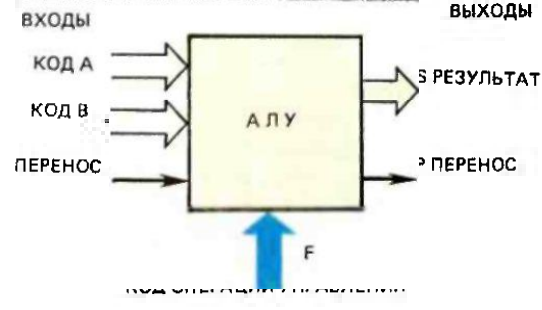
Рис. 6.16. Мультиплексор-демультиплексор.

плексора. Принцип действия мультиплексора поясняется рисунком 6.16. Предполагается, что исходные состояния счетчиков одинаковы, а импульсы управления строго синхронны. В вычислительных устройствах такие импульсы вырабатываются единым для всей системы тактовым генератором.

6.7. Как считает ЭВМ?

Обработка информации на ЭВМ сводится к процедурам, которые умеет выполнять процессор — к логическим и арифметическим операциям. Используя, например, представление чисел в дополнительном коде (см. 3.10), вычитание, умножение и деление можно свести к действиям одного типа — суммированию. Сочета-

Рис. 6.17. Сигналы арифметико-логического устройства.



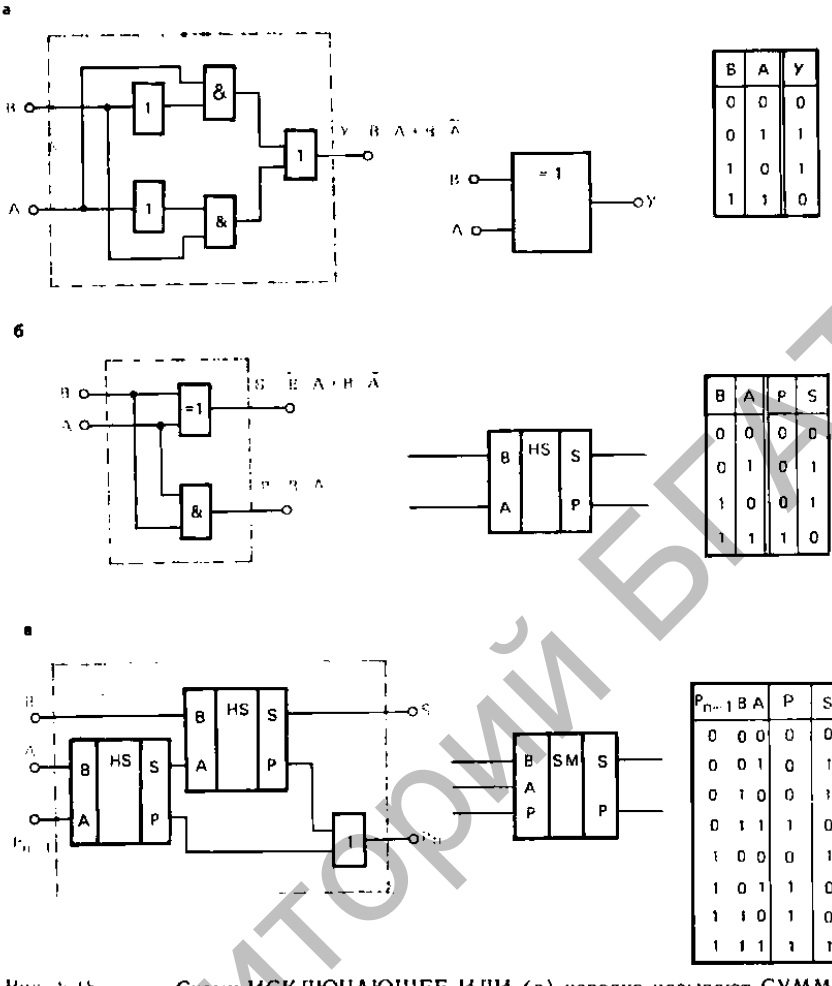


Рис. 6.18. Схему ИСКЛЮЧАЮЩЕЕ ИЛИ *(а)* нередко называют СУММАТОРОМ ПО МОДУЛЮ ДВА, или СХЕМОЙ НЕРАВНОЗНАЧНОСТИ. Она является составным элементом полусумматора *(б)*. Сумматор *(в)* традиционно представляют состоящим из двух полусумматоров.

ние суммирования со сдвигом (см. 6.5) позволяет осуществить умножение n-разрядных двоичных чисел за n операций суммирования и $(n-1)$ операций сдвига. Деление опять же сводится к суммированию и сдвигу, а число этих операций также сравнительно невелико и зависит от требующейся точности.

Часть процессора, в которой выполняются операции над кодами чисел, называют *арифметико-логическим устройством* (АЛУ). Остальные его составляющие относятся к *устройству управления* (УУ). Одна из простейших задач УУ — подавать в АЛУ оба операнда и шифр операции, выполняемой над ними.

В простейшем случае АЛУ представляет собой систему комби-

национных схем, входами которых являются коды чисел A и B и сигнал переноса C. На выходе АЛУ получают код результата и сигнал переноса P. Вид операции, выполняемой над числами A и B, определяет код управления F (рис. 6.17).

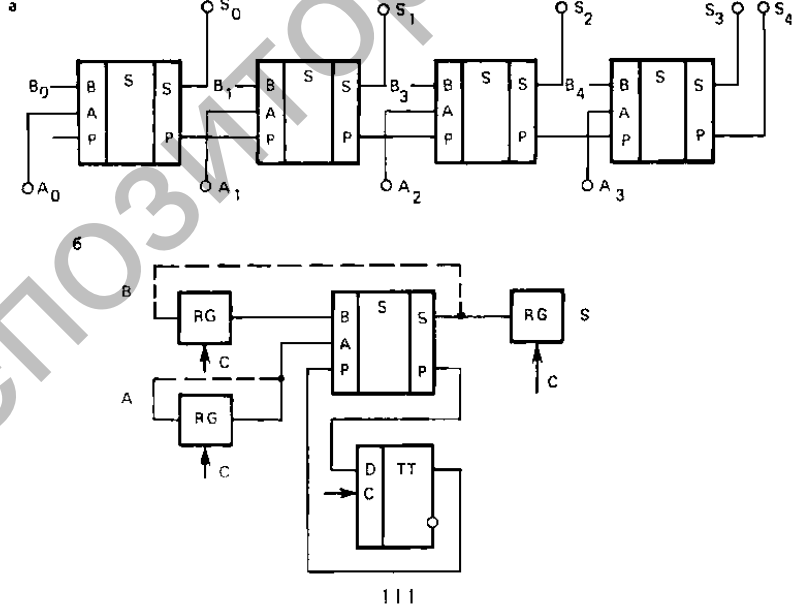
В выпускаемых промышленностью микросхемах АЛУ коды A, B и F — четырехразрядные. Несколько таких АЛУ могут быть соединены для параллельной обработки чисел большей длины.

В режиме арифметического суммирования АЛУ функционирует как четырехразрядный *сумматор*, который как и любой параллельный многоразрядный *сумматор* вообще составлен из одноразрядных сумматоров. Каждая из его составляющих должна уметь складывать три одноразрядных числа, так как к суммируемым цифрам данного разряда может прибавляться еще и цифра переноса из младшего разряда.

Рисунок 6.18 поясняет действие одноразрядного сумматора и его составляющих, а рисунок 6.19 — методы построения параллельного и последовательного сумматоров многоразрядных чисел.

В современных ЭВМ умножение производится в одно действие при помощи так называемых матричных умножителей.

Рис. 6.19. Устройства для суммирования многоразрядных двоичных чисел: параллельный сумматор *(а)* и сумматор последовательного типа *(б)*. Во втором случае C-входы всех m-разрядных сдвигающих регистров и D-триггера соединены между собой. После поступления на этот общий C-вход m импульсов в регистре будет сумма чисел $B + A$. Пунктиром показаны соединения, позволяющие занести сумму в регистр A (т. е. превратить его в аккумулятор и сохранить слагаемое в регистре B (сравните с рис. 6.9).



6.8. Электронная память, или СОЗУ, ОЗУ, ПЗУ, ППЗУ и т. п.

В любой ЭВМ имеется *запоминающее устройство* (ЗУ). Емкость,или объем ЗУ — это количество информации, которое может храниться в нем в одно и то же время. Поэтому объем ЗУ измеряется в тех же единицах, что и количество информации — в битах (1 или 0), байтах (1 байт = 8 бит) или единицах К (сокращение от «кило», но $1К = 2^{10} = 1024$, а не 1000) и М ($1М = 2^{20}$).

Большинство ЗУ организовано таким образом, что в них хранят двоичные коды определенной разрядности. Такой код размещают во вполне определенном месте ЗУ. Каждому месту присваивается номер, или *адрес*. Адрес, выраженный в двоичной системе счисления, тоже является двоичным кодом, поэтому адреса также могут храниться в ЗУ (см. 2.10).

ЗУ в целом выполняет две основные операции: заносит слово по указанному адресу **(запись)** и сообщает, какое слово хранится по заданному адресу **(считывание).** Длительности этих процедур — время записи и время считывания — определяют быстродействие ЗУ, а большее из них называют *временем обращения к ЗУ*. В полупроводниковых ЗУ время обращения, как правило, меньше одной миллионной доли секунды (1 мкс).

Наибольшим быстродействием характеризуется СОЗУ. За ним следует ОЗУ. В СОЗУ и ОЗУ информация может как записываться, так и считываться в процессе вычислений и обработки данных.

Для хранения данных, которые в процессе работы ЭВМ не изменяются, служат ПЗУ и ППЗУ — *постоянные и программируемые постоянные ЗУ*. Из ПЗУ и ППЗУ информация только считывается, причем в ПЗУ она может быть занесена лишь один раз (при изготовлении), а в ППЗУ записывается, стирается и перезаписывается на специальном стенде. ППЗУ называют иногда *репрограммируемыми ЗУ*.

Электронная память типа ОЗУ, СОЗУ, ПЗУ, ППЗУ организована таким образом, что в произвольный момент времени можно обратиться к любой ячейке, т. е. считать или записать код по произвольному адресу. Поэтому их называют ЗУПВ.

В отличие от ЗУПВ в *ЗУ на магнитных дисках*, магнитных лентах, ультразвуковых линиях задержки и др. информация записана последовательно код за кодом, и произвольный доступ к адресам таких ЗУ невозможен. Электронная память также может иметь последовательную структуру. Различают два способа ее организации: «первым вошел — последним вышел», или *стек*, и «первым вошел — первым вышел», или *очередь* (FIFO).

В основу работы современных ЗУ положены различные физические явления: магнитные, электрические, ультразвуковые, оптические, включая голографические и многие другие. Однако в настоящее время в мини-ЭВМ и микропроцессорных системах преобладают полупроводниковые СОЗУ, ОЗУ, ПЗУ, ППЗУ, а в качестве ЗУ большой емкости — устройства хранения информации на магнитных дисках и лентах.

Полупроводниковые ОЗУ, ПЗУ и ППЗУ — это БИС с наиболее высокой степенью интеграции. Если еще недавно к ординарным относили ЗУ в виде отдельной микросхемы емкостью 256 бит, то сейчас ординарной стала ИС ЗУ на 256 Кбит и ожидается появление микросхем ОЗУ емкостью 1, 4 и 16 Мбит.

6.9. Магистраль

На определенном этапе развития вычислительной техники добавление в структуру ЭВМ новых блоков порождало все увеличивающееся количество проводов для связи этих блоков с остальными устройствами ЭВМ. Но в начале 60-х годов был обоснован принцип и построены первые ЭВМ, в которых дополнительные устройства требовали не введения новых линий связи, а лишь удлинения уже существующих, причем такое удлинение конструкторы, как правило, предусматривали при разработке ЭВМ, устанавливая в них «лишние» разъемы для дополнительных блоков.

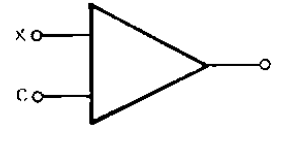
Новый способ обмена информацией сильно повлиял на структуру самих ЭВМ и в значительной мере способствовал появлению микропроцессоров. Обмен между многими источниками и приемниками стали осуществлять по группе одних и тех же проводов *(шин)*, названных *магистралью.*

Подача сигнала на любой из проводов магистрали осуществляется единообразно через так называемые *магистральные усилители* («тристабильные схемы»).

Магистральный усилитель имеет два входа — информационный *X* и вход управления *C*. Если на входе *C* находится логическая 1, то на выходе *Y* повторяется сигнал *X*. Если же на *C* ноль, то независимо от сигнала на входе *X* выход *Y* изолирован и не оказывает никакого влияния на шину, к которой он присоединен (рис. 6.20).

Источниками кодов, подаваемых в магистраль, являются регистры, размещенные в различных блоках. Выходы триггеров каждого такого регистра выводятся на общую шину через группу магистральных усилителей с объединенными *C*-входами (рис. 6.21). Сигналы разрешения на такие общие *C*-входы различных блоков вырабатываются схемой типа дешифратора,

Рис. 6.20. Обозначение и описание работы магистрального усилителя.



C	X	Y
0	0	$R_{вых} = \infty$
0	1	
1	0	0
1	1	1

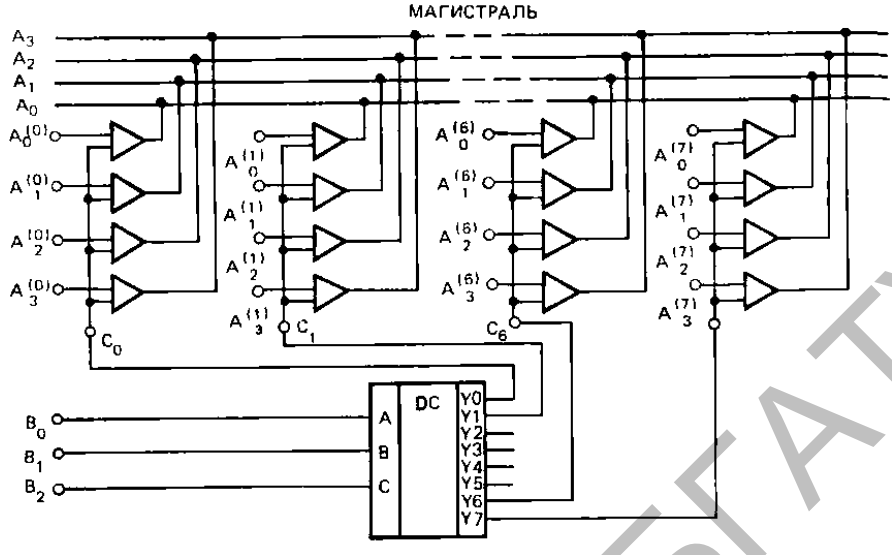


Рис. 6.21. В каждый момент времени логическая единица наблюдается лишь на одном из выходов дешифратора, поэтому информация на шины магистрали поступает только из одного источника.

поэтому одновременное подключение к магистрали двух и более источников исключено. Однако показанный на рисунке 6.21 дешифратор обычно не существует в виде автономного устройства, а как бы расчленен на части, находящиеся в блоках. Каждая такая часть дешифрирует одно значение кода на шинах B (на рис. 6.21 — B_0, B_1, B_2), которые разводятся таким же образом, как и остальные провода магистрали, но в их функции не входит прием информации от различных блоков. По ним во все блоки передаются коды управления, которые в самих блоках преобразуются в управляющие сигналы. Подобным образом в блоки, подключенные к магистрали, могут поступать и другие коды управления.

6.10. Окно в аналоговый мир или ЦАП и АЦП

Для имитации сложных сигналов управления исполнительными механизмами, наглядного представления итогов обработки данных и многих других целей широко используют *цифро-аналоговые преобразователи* (ЦАП) — устройства, превращающие код в пропорциональное ему напряжение или ток. Чаще всего преобразованию подвергается параллельный двоичный код связанного с ЦАП триггерного регистра, а действие ЦАП основано на суммировании токов (или напряжений), пропорциональных весам триггеров, находящихся в состоянии 1 (рис. 6.22). Разрядность современных ЦАП достигает 16, что эквивалентно возможности вырабатывать до $2^{16}=65\,536$ градаций напряжения.

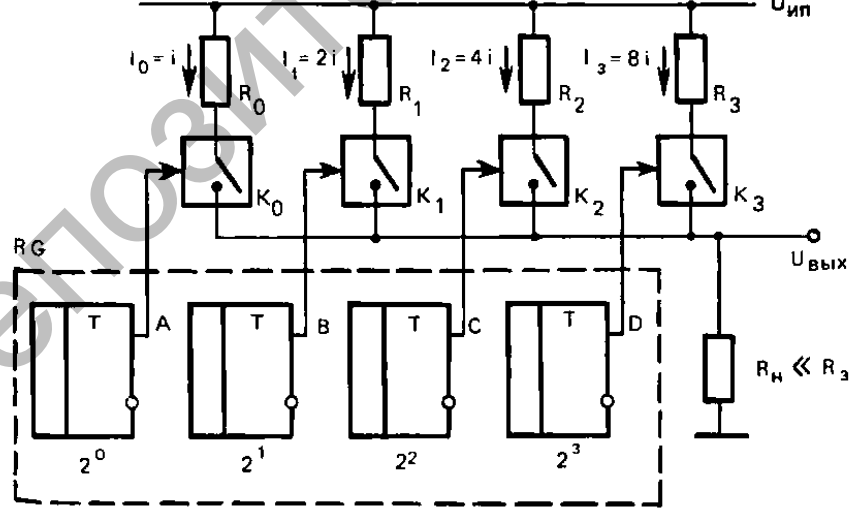
Цифровые значения различных физических величин получают при помощи *аналого-цифровых преобразователей* (АЦП). Под цифрой понимают обычно параллельный двоичный (двоично-десятичный) код, т. е. сигнал, пригодный для непосредственной обработки устройствами вычислительной техники.

На вход АЦП поступают электрические сигналы, чаще всего — напряжение. Нередко их источниками служат преобразователи различных физических величин в их электрические аналоги. Сочетание преобразователей неэлектрических величин в их электрические аналоги и АЦП позволяет придавать любым физическим величинам форму, необходимую для обработки на ЭВМ.

В общем случае АЦП, преобразующий в код электрическое напряжение, состоит из двух частей — *устройства выборки и хранения* (УВХ) и собственно АЦП. УВХ запоминает напряжение в нужный момент времени и удерживает его неизменным до тех пор, пока собственно АЦП не осуществит преобразование в код. Запоминающим элементом УВХ служит конденсатор. При таком подходе можно считать, что собственно АЦП преобразует в код п о с т о я н н о е напряжение. Сравнение этого постоянного напряжения с напряжением, цифровое значение которого известно,— один из способов аналого-цифрового преобразования.

Источником напряжения с известным цифровым эквивалентом может быть ЦАП типа изображенного на рисунке 6.22 с высокостабильным источником питания. Если удается подобрать код на

Рис. 6.22. **ЦАП. Управляя замыканием ключей, триггеры включают в общую нагрузку $R_н$ токи, пропорциональные весу разрядов, поэтому падение напряжения на резисторе $R_н$ пропорционально коду DCBA. В многоразрядных ЦАП токи задаются не резисторами, а электронными стабилизаторами тока и преобразуются в величину, пропорциональную весу разряда, при помощи делителей на прецизионных резисторах.**



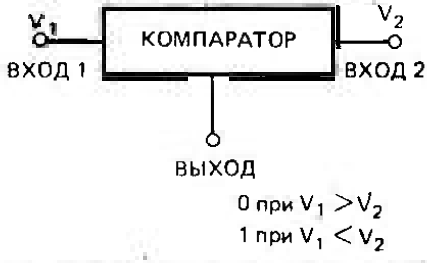


Рис. 6.23. Сигналы компаратора.

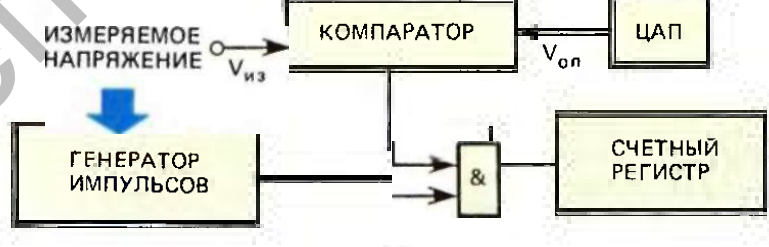
входе ЦАП, при котором его выходное напряжение равно измеряемому, этот код и есть цифровой эквивалент измеряемого напряжения.

В зависимости от алгоритма нахождения искомого кода различают два основных типа АЦП — с *единичным приращением* и *поразрядным уравновешиванием*.

Работа первого из них напоминает процесс взвешивания, при котором на одну чашу весов кладут груз, «равный» постоянному напряжению на выходе УВХ, на вторую — одну за другой гирьки единичной массы. Если при очередном добавлении чаша с гирями перетянет, то число гирь на ней — искомый цифровой эквивалент измеряемого напряжения. Роль «электронных весов» выполняет весьма распространенная схема — *компаратор* (от. лат. *comparator* — сравниваю). Она имеет два входа и один выход. Если напряжение на первом входе меньше, чем на втором, на выходе — логическая 1, если же наоборот — на выходе 0 (рис. 6.23). В качестве чаши, наполняемой гирями, служит счетчик импульсов на триггерах. Рисунок 6.24 поясняет действие АЦП с единичными приращениями.

При поразрядном уравновешивании структура АЦП остается той же, но используются не единичные уравновешивающие напряжения («гири»), а напряжения с весами 1, 2, 4, ..., 2^{n-1}. Сначала включается напряжение 2^{n-1}, затем к нему добавляется 2^{n-2}

Рис. 6.24. АЦП. При открытой схеме И каждый импульс генератора увеличивает содержимое счетчика на 1 и напряжение на выходе ЦАП получает единичное приращение. Когда это напряжение сравняется с измеряемым, компаратор запретит поступление импульсов в счетный регистр, а установившийся в нем код будет цифровым эквивалентом $V_{из}$. После считывания этого кода счетчик сбрасывается в 0 и АЦП готов к следующему измерению.



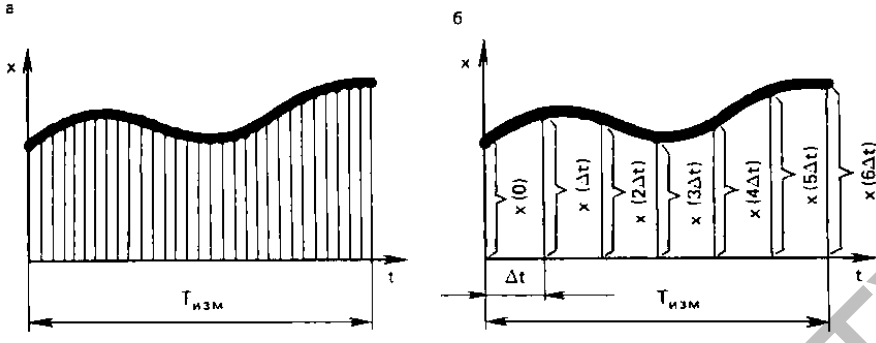


Рис. 6.25. Чисто интуитивно для замены набором цифр непрерывной функции $x(t)$ отсчеты должны сниматься через как можно меньшие промежутки времени *(а)*. Практически *(б)* достаточно измерить («оцифровать») значения функции через интервалы времени $\Delta t = 1/2f_c$, где f_c — предельная частота в спектре функции $x(t)$. Как правило, f_c известна или же может быть оценена в каждом конкретном случае.

и т. д. до 1. После каждого добавления анализируется сигнал компаратора. Если компенсирующее напряжение превысило измеряемое, последнюю добавку отключают, сбрасывая триггер соответствующего разряда, если же нет — оставляют и т. д. Искомый цифровой эквивалент находят за n процедур сравнения (в отличие от 2^n при методе единичных приращений), что и обуславливает более высокое быстродействие АЦП с поразрядным уравновешиванием.

Высокое быстродействие АЦП позволяет вводить в ЭВМ информацию о высокоскоростных процессах, так как заменить непрерывно изменяющийся сигнал набором цифр, например набором мгновенных значений этого сигнала, можно только в том случае, если набор цифр несет такую же информацию, как и сам сигнал. Ответ на вопрос, с какой минимальной частотой нужно преобразовывать мгновенные значения сигнала в цифру, дает теорема Котельникова, суть которой поясняется рисунком 6.25.

6.11. Еще раз о кодах

Предположим, что необходимо вводить в ЭВМ информацию о линейных перемещениях. Простейший способ — воспользоваться линейкой с делениями и преобразовывать соответствующие десятичные цифры в двоичный код. Если же линейка проградуирована в двоичной системе счисления, то отсчеты можно сразу получать в виде кода.

Чтобы автоматически следить за перемещениями, на линейку наносят проводящее покрытие, как показано на рисунке 6.26, *а*. Если к нему подвести электрическое напряжение, соответствующее логической 1, то со скользящих контактов, расположенных

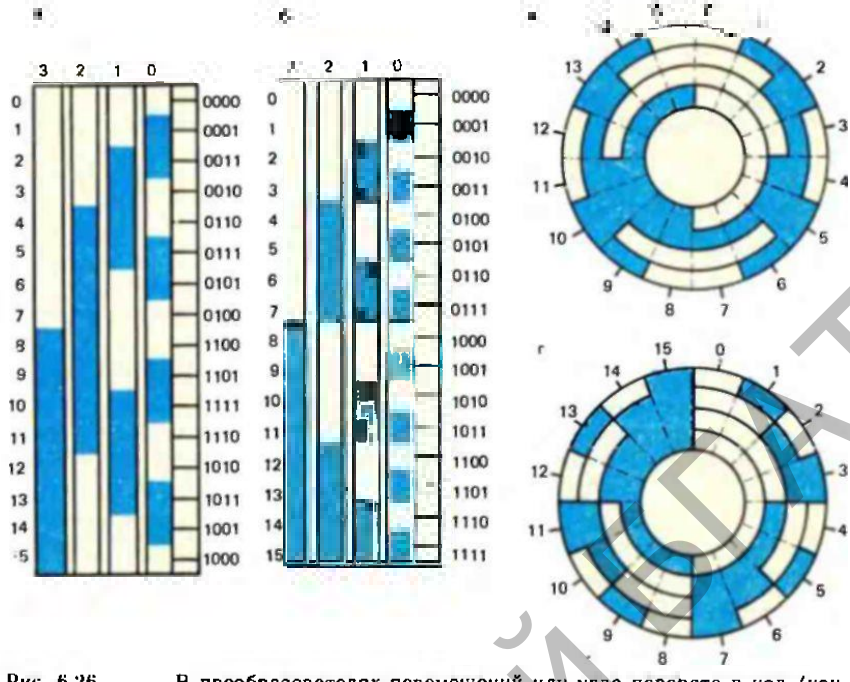


Рис. 6.26. В преобразователях перемещений или угла поворота в код (контактных и с фотосчитывателем) шкалы градуируются в коде Грея *(а, в)*, и ошибка считывания не превышает одной наименьшей градации. В случае обыкновенного двоичного кода *(б, г)* она может достигать единицы в любом разряде.

на линии, перпендикулярной к длинной стороне линейки, будут сниматься коды, пропорциональные расстоянию от ее начала. Аналогичным образом в код может быть преобразован угол поворота (рис. 6.26, *а, в*). Однако и в том, и в другом случае применение прямого двоичного кода таит опасность получить большие ошибки считывания, обусловленные неточностью нанесения проводящего контакта и неточностью расположения скользящих контактов.

К сожалению, подобного рода неточности всегда имеют место и избежать их невозможно, поэтому если, например, при переходе от отсчета 7 к отсчету 8 контакт старшего разряда соприкоснется с проводящим слоем раньше, чем разомкнутся контакты младших разрядов, то вместо кода числа 8 будет считан код числа 15, и ошибка измерения длины составит почти 50 %.

Для исключения подобного рода грубых ошибок применяются специальные двоичные коды. Один из них — *код Грея* — характеризуется тем, что при переходе от любого числа к соседнему в коде изменяется только одна значащая цифра. Принцип построения преобразователей в коде Грея поясняет рисунок 6.26, *б, г*.

Код Грея получается из прямого двоичного кода при помощи одной операции — суммирования по модулю 2. Число в прямом

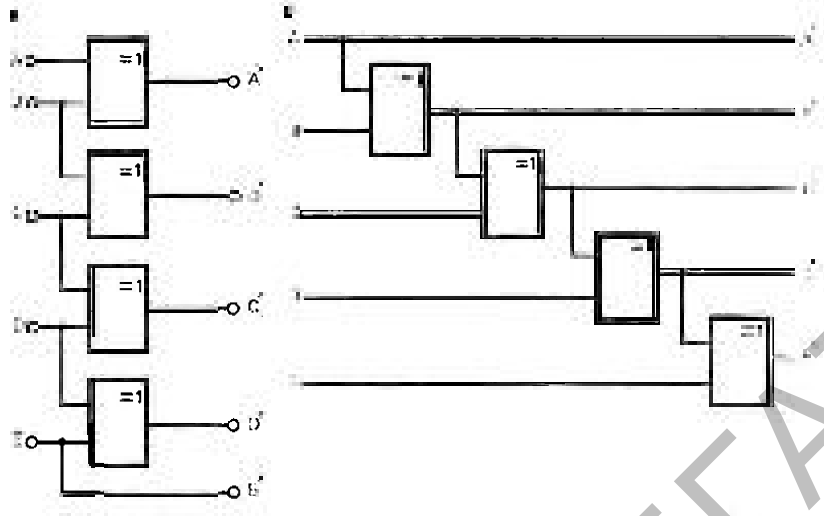


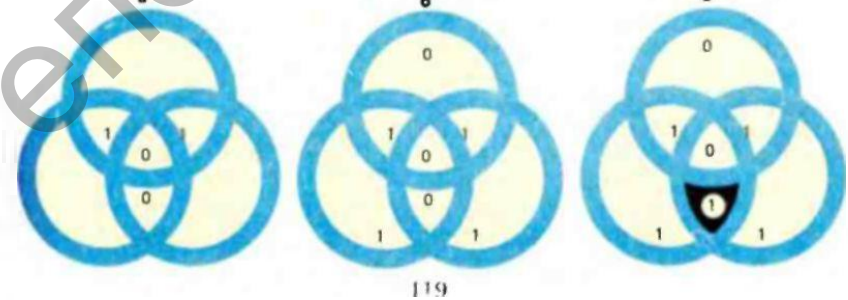
Рис. 6.27. Схемы преобразования кодов: прямого — в код Грея *(а)* и кода Грея — в прямой *(б)*.

двоичном коде $EDCBA$ преобразуется в число в коде Грея $E'D'C'B'A'$ по следующим правилам:

$$\begin{aligned} A' &= A \oplus B, \\ B' &= B \oplus C, \\ C' &= C \oplus D, \\ D' &= D \oplus E, \\ E' &= E \oplus 0 = E, \end{aligned} \tag{6.1}$$

где знаком $\oplus$ обозначена сумма по модулю 2 (см. гл. 4). На рисунке 6.27 пояснен принцип построения преобразователей двоичного кода в код Грея и наоборот.

Рис. 6.28. Диаграммы, поясняющие принцип формирования кода с исправлением ошибок. Подлежащее передаче слово 1010 *(а)* дополняется кодом 101 *(б)* таким образом, чтобы сумма цифр внутри каждого круга была четной. Если в процессе передачи-приема один из символов идентифицирован ошибочно *(в)*, то это легко выявить и исправить, так как четность нарушится в двух кругах, а ошибочный символ находится в зоне их перекрытия.



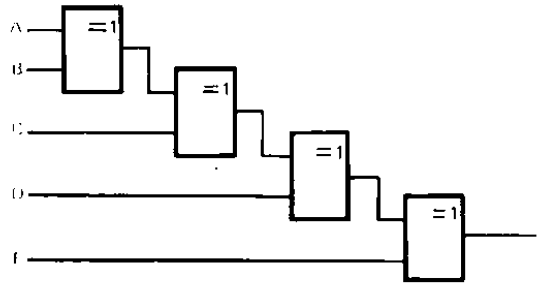
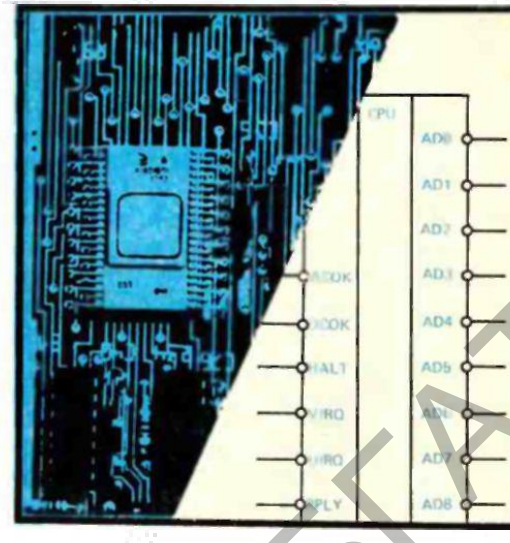


Рис. 6.29. Принцип построения многовходового устройства проверки на четность.

При передаче двоичных кодов по каналам связи на большие расстояния или же в условиях помех сигналы, соответствующие отдельным символам, могут исказиться, и приемник идентифицирует их неправильно. Подобного рода ошибки могут привести к крайне нежелательным последствиям. Избежать их позволяют коды с исправлением ошибок, часто называемые *кодами Хемминга* по имени инженера, предложившего их в конце 40-х годов.

Для этого по линии связи наряду с подлежащим передаче кодом передаются дополнительные символы. Их значения, т. е. 1 или 0, выбираются такими, чтобы сумма цифр определенных позиций всегда была четной. В приемнике производится проверка этих же сумм на четность. Сочетания позиций выбираются таким образом, чтобы по результатам проверки на четность можно было точно установить, какой именно бит принят ошибочно. При этом предполагается, что ошибка появляется не более, чем в одной позиции (рис. 6.28). Если же возможен ошибочный прием двух битов, то, в принципе, и это можно и установить, и исправить, но при помощи более сложных кодов.

Заметим, что проверка на четность производится при помощи схемы ИСКЛЮЧАЮЩЕЕ ИЛИ, причем число входов устройства проверки на четность увеличивается таким же способом, как и число входов схем И и ИЛИ (рис. 6.29).



7.1. Микропроцессор [illegible] ЭВМ
7.2 [illegible]

МИКРОПРОЦЕССОР КАК ИНТЕГРАЛЬНАЯ СХЕМА

Созданный в 1971 году первый микропроцессор (МП) был предназначен для работы в качестве процессора четырехразрядной ЭВМ и размещался в стандартном корпусе с шестнадцатью выводами. Собственно микросхема была выполнена на кремниевой пластинке размерами 2,8×3,8 мм и содержала 2250 транзисторов.

В вычислительной технике МП — «кирпичик» для построения компактных высокопроизводительных ЭВМ, в электронике — основной элемент **контро́ллеров** — устройств управления, непосредственно встраиваемых в объекты.

И в том, и в другом случае основная функция МП — выполнение написанных для него программ, что возможно только при постоянном взаимодействии МП с окружающим миром. Микропроцессор должен откуда-то получать данные и куда-то направлять результаты их обработки. Кроме того, само выполнение программы требует поступления в МП кодов команд из ОЗУ или ПЗУ, которые обычно не входят в его состав.

Использованные в данной и последующих главах названия сигналов, информационных линий, аббревиатуры команд имеют англоязычное происхождение.

7.1. Микропроцессор — еще не ЭВМ

Двигатель может выполнять различную работу, но требует для этого источника энергии, приспособлений, формирующих движения нужного характера, и объектов воздействия. Работа МП возможна при наличии источников электропитания и, по

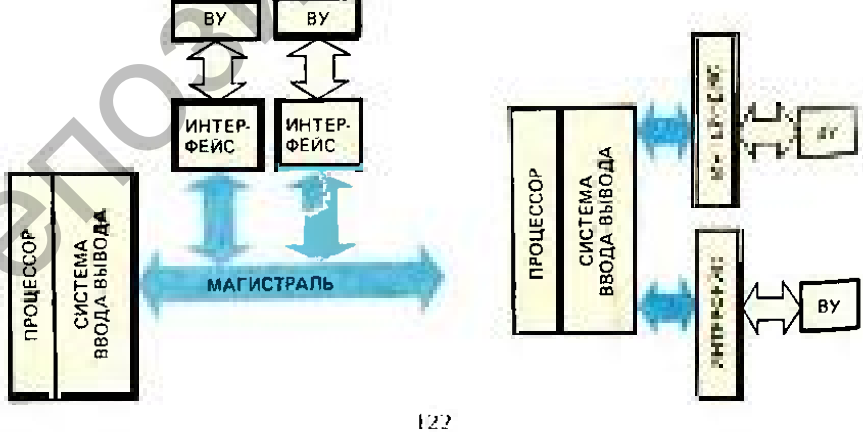
крайней мере, двух типов устройств: запоминающих и *ввода-вывода данных* (УВВ). Связь МП с УВВ и ЗУ осуществляется по *шинам* — системам проводов, несущих сигналы от МП к *внешним устройствам* (ВУ) и обратно. Выработка и прием этих сигналов как в МП, так и в ВУ осуществляется специальными схемами, называемыми интерфейсными. Вообще *интерфейс* (от англ. *interface* — стык, взаимодействие) — это совокупность средств, определяющих логический порядок взаимодействия систем (протокол) и вытекающие из протокола требования к аппаратуре и к программному обеспечению, если обмен данными осуществляется под управлением программы.

На практике под интерфейсом чаще всего понимают только электрическую схему, содержащуюся в устройстве для сопряжения его с ЭВМ, а интерфейсные средства, входящие в состав УУ процессора, носят название *системы ввода-вывода.* Чтобы упростить интерфейс внешних устройств, самые сложные его функции возлагают на систему ввода-вывода процессора. При наиболее распространенном (магистральном) принципе организации интерфейсы устройств единообразно подключаются к общей системе шин — *магистрали*, исходящей из процессора (рис. 7.1, *а*). При *радиальной организации интерфейса* к каждому устройству от процессора идет своя система шин (рис. 7.1, *б*).

Состав ВУ, подключаемых к магистрали МП, зависит от характера решаемых задач. Рассмотрим три основные конфигурации систем на базе МП (рис. 7.2).

Контроллер, или специализированное устройство обработки информации (рис. 7.2, *а*), помимо МП, содержит ПЗУ для хранения заранее отлаженной программы;

Рис. 7.1. В вычислительных системах используются магистральный (*а*) и радиальный (*б*) принципы организации интерфейса. Физически магистраль выполняется в виде многопроводной шины с разъемами для подключения внешних устройств (ВУ), а в одноплатной микро-ЭВМ — с гнездами (панельками) для микросхем.



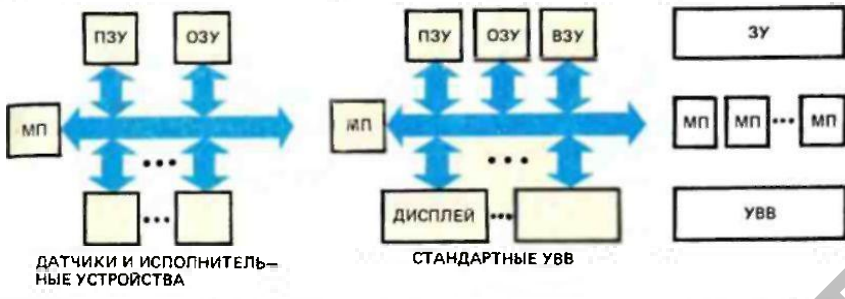


Рис. 7.2. Основные конфигурации систем на основе микропроцессоров: контроллер, или специализированная микро-ЭВМ *(а)*, микро-ЭВМ общего назначения *(б)*, многопроцессорная ЭВМ, или система распределенной обработки информации *(в)*. Для каждой из систем последнего типа внутренние связи специфичны. Интерфейсные схемы не показаны.

ОЗУ для входных, промежуточных или выходных данных, в качестве которого в простейших случаях служат имеющиеся в МП РОНы; интерфейсы используемых датчиков и исполнительных устройств, нередко выполняемые в виде специальных БИС.

Микро-ЭВМ общего назначения (рис. 7.2, *б*) содержит МП, достаточно емкое ОЗУ, ПЗУ для хранения часто используемых программ, в том числе программ для начальной загрузки. В итоге этой загрузки в ОЗУ помещается программа, обеспечивающая диалог ЭВМ с человеком. Диалог осуществляется обычно посредством терминала — стандартного внешнего устройства, например алфавитно-цифрового дисплея, в котором ввод символов производится с клавиатуры, а вывод — на экран электронно-лучевой трубки. Перечисленные устройства образуют ЭВМ так называемой минимальной конфигурации. Она может быть расширена путем подключения внешних ЗУ (ВЗУ) типа *накопителей на магнитных дисках* (НМД), на *гибких магнитных дисках* (НГМД), на *магнитных лентах* (НМЛ), печатающих устройств, графопостроителей, средств телеобработки, обеспечивающих связь ЭВМ с удаленными объектами, в том числе и с другими ЭВМ (модемы, адаптеры, мультиплексоры передачи данных).

Интерфейсы некоторых из перечисленных устройств выполняют достаточно сложные функции и фактически представляют собой контроллеры. Нередко они строятся на основе собственных микропроцессоров. По этой причине основной МП микро-ЭВМ будем называть в дальнейшем *центральным процессором* (ЦП).

МП как составная часть высокопроизводительной ЭВМ или системы распределенной обработки данных (рис. 7.3, *в*). В простейших случаях такие важные параметры как длина машинного слова могут быть пропорциональны числу используемых МП. С его увеличением фактически возрастает параллелизм, а значит,— быстродействие обработки

информации. В иных случаях отдельные МП решают различные, независимые друг от друга части сложной задачи. Системы такого типа носят название *мультипроцессорных* или параллельных. Конечно, параллельный характер работы для них следует понимать в более широком смысле, чем параллельную, но одинаковую обработку битов в многоразрядном слове.

7.2. Типы микропроцессоров

По характеру использования различают три основных типа МП.

1. *Однокристальные микро-ЭВМ* (микроконтроллеры), предназначенные для встраивания их в приборы, станки и другое оборудование. С целью минимизации дополнительно подключаемых ИС для таких МП характерно наличие на одном кристалле хотя бы небольших по объему ОЗУ и ПЗУ, нескольких распространенных интерфейсов ввода-вывода информации, в том числе и в аналоговой форме.

2. *МП общего назначения* чаще всего применяются в микро-ЭВМ. Они характеризуются стремлением разработчиков добиться максимального быстродействия при как можно больших длине слова и объеме адресуемой памяти и развитой системой команд, удобной для решения самых разнообразных задач. Такие МП могут быть реализованы в виде одной или нескольких БИС, образующих т. н. базовый микропроцессорный комплект. Каждая из ИС этого комплекта реализует отдельный узел или функциональный блок МП, например АЛУ и УУ.

3. *Секционированные МП*, позволяющие наращивать длину обрабатываемого слова увеличением числа используемых секций, имеют максимальное быстродействие, используются для построения высокопроизводительных ЭВМ и быстродействующих специализированных вычислительных систем.

При характеристике конкретного МП необходимо указать его принадлежность к одному из рассмотренных типов. При этом обычно выделяют следующие параметры:

1) длину слова (разрядность);

2) объем адресного пространства — максимально возможный диапазон адресов ОЗУ;

3) особенности системы ввода-вывода и магистрали;

4) количество и функции РОН;

5) способы адресации памяти;

6) систему команд и систему микропрограммирования (см. 8.10), если оно возможно;

7) быстродействие, обычно оцениваемое количеством операций определенного типа, выполняемых за одну секунду.

Далее будет рассмотрен универсальный шестнадцатиразрядный однокристальный микропроцессор К1801ВМ1, имеющий быстродействие около 500 тысяч операций в секунду для регистровых команд, который используется в микро-ЭВМ «Электроника

МС1201», входящих в состав диалоговых вычислительных комплексов типа ДВК-1, ДВК-2 и др. Такой выбор не случаен, так как эта ЭВМ — одна из представителей многочисленного семейства машин, чрезвычайно широко распространенных как в нашей стране, так и за рубежом. Все ЭВМ этого семейства совместимы по своему программному обеспечению и отличаются в основном конфигурацией и быстродействием. В их числе: микро-ЭВМ «Электроника-60», «Электроника-81», «Электроника-82», «Электроника-85», машины на основе микропроцессорного комплекта К588, мини-ЭВМ СМ-3, СМ-4, СМ-1420, СМ-1300, СМ-1600, «Электроника-100/25», «Электроника-79» и др. модели. За рубежом машины этого семейства наиболее широко представлены моделями серий PDP-11 (мини-ЭВМ) и LSI-11 (микро-ЭВМ) фирмы DEC. Высокие технические характеристики и надежность перечисленных машин сделали их общепринятым стандартом для решения самых разнообразных задач.

7.3. Способы обмена данными

Представьте себе, что, поставив на плиту чайник, вы удобно уселись в соседней комнате с увлекательной книгой. Маловероятно, что чайник вскипит в тот самый момент, когда, дойдя до конца главы, можно будет без большого сожаления остановиться. Скорее всего, придется периодически отвлекаться, чтобы присматривать за чайником. В более выгодном положении оказывается владелец чайника со свистком, сигнализирующим о закипании воды.

Пример имеет вполне определенное отношение к способам обмена данными между процессором и внешними устройствами.

Как правило, все ВУ работают несинхронно с процессором и, за исключением ОЗУ, значительно медленнее его. Поэтому и для процессора, и для ВУ сигналы магистрали появляются в непредсказуемые моменты времени. Чтобы исключить потери информации в процессе обмена, необходимы специальные меры.

Интерфейс каждого ВУ имеет как минимум два специальных программно-доступных регистра. Один из них — *регистр данных* (РД) — служит в качестве буфера между магистралью и ВУ. Он хранит слово, выработанное в ВУ, до того момента, когда процессор будет в состоянии принять его. В операциях вывода процессор помещает данные в РД, где они могут храниться до момента высвобождения ВУ. В обоих случаях и процессор, и ВУ, передав слово в РД, могут незамедлительно продолжить свою работу, но к факту помещения слова в РД или готовности к его приему должно быть привлечено внимание процессора. Это осуществляется с помощью второго регистра — *регистра состояния* (РС).

В простейшем случае используется только один бит РС, называемый *флагом готовности*. Этот бит устанавливается в единичное состояние, если РД готов к очередному сеансу обмена. И РД, и РС подключены к магистрали, поэтому процессор может определить готовность ВУ при проверке содержимого РС. Если флаг уста-

новлен, производится условный переход к команде считывания или записи, после выполнения которой флаг автоматически сбрасывается.

При неустановленном флаге ВУ процессор циклически продолжает его опрос. Такой способ обмена данными процессора с ВУ называется *асинхронным программным обменом* (рис. 7.4).

Для обмена со сложными ВУ процессору может оказаться недостаточно информации только о готовности ВУ, поэтому состояние ВУ отражается несколькими битами РС.

Основной недостаток асинхронного обмена — бесполезные потери времени процессора на ожидание готовности ВУ. Значительно уменьшить эти потери позволяют *прерывания программы*. В этом случае УУ процессора через специальную линию магистрали реагирует на установку флага ВУ автоматическим переключением с исполняемой программы на заранее подготовленную программу, которая и производит обмен данными.

С реализацией прерываний связано несколько трудностей. Во-первых, по завершению программы обслуживания ВУ необходимо продолжить выполнение прерванной программы, полностью восстановив состояние процессора к моменту прерывания. Во-вторых, при наличии нескольких источников прерываний, подключенных к единственной линии запроса, процессору нужно уметь быстро «распознать» ВУ, нуждающееся в обслуживании. В-третьих, нужно эффективно разрешать конфликтные ситуации, когда, например, прерывания требуют одновременно несколько ВУ, либо новый сигнал прерывания (запрос) приходит в момент, когда еще не кончилась обработка предыдущего запроса *(«вложенные» прерывания)*.

Распространенный способ преодоления этих трудностей – *векторная система прерываний*. При этом программы обслуживания ВУ организуются так же, как обычные *подпрограммы**, с той разницей, что их вызов осуществляется не специальной командой, а аппаратно, по запросам с магистрали. Как при обращении к подпрограммам, так и в момент прерывания УУ автоматически заносит содержимое счетчика команд и ССП в стек. Помимо сигнала запроса, ВУ обязано установить на шинах магистрали адрес вектора прерывания в ОЗУ. *Вектор прерывания* — две последовательные ячейки памяти, в которых находятся начальные адрес и состояние подпрограммы обслуживания данного ВУ. Получив адрес вектора прерывания, процессор автоматически загружает СК и ССП содержимым ячеек, образующих вектор, что обеспечивает выполнение подпрограммы обслуживания. В ее конце обязательно должна присутствовать специальная команда возврата из подпрограммы, по которой из стека извлекаются «старые» значения СК и ССП, обеспечивающие продолжение выполнения основной программы с того места, где она была

* Подпрограмма — фрагмент программы, допускающий многократное обращение из различных мест основной программы и обеспечивающий автоматический возврат в то место, откуда произошло обращение (подробнее см. 8.4).

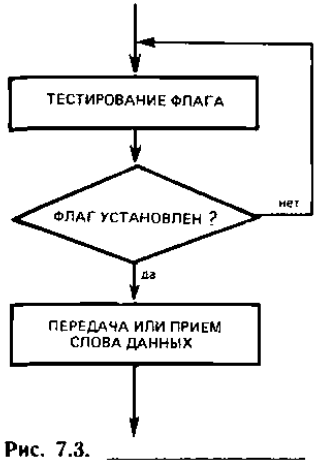


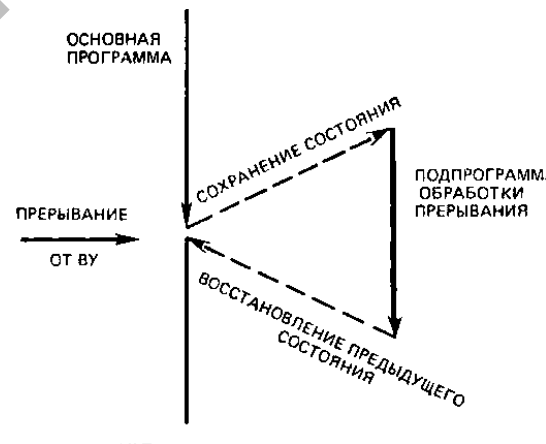
Рис. 7.3.
Программа для асинхронного обмена данными содержит команду тестирования флага, т. е. считывания его состояния в один из битов ССП. Следующая команда — условная передача управления — заставляет процессор вновь повторять тестирование, если принятое состояние флага — нулевое.

прервана (рис. 7.4). Основной принцип стека «первым вошел — последним вышел» обеспечивает правильную последовательность возврата в основную программу в случае вложенных прерываний.

Единичное значение специального бита разрешения прерывания в ССП блокирует всю систему прерываний, а соответствующие комбинации битов в РС запрещают выдачу на магистраль запросов на прерывание от отдельных устройств или, что то же самое, осуществляют *маскирование прерываний*.

Гибкость системы прерываний может быть повышена назначением *приоритетов* как основной программе, так и подпрограммам обслуживания ВУ. С этой целью используются выделенные группы битов в ССП и векторах прерываний. В «старших» ЭВМ упомянутого семейства допустимы 4 уровня приоритета (от 4-го до 7-го). Получив запрос на прерывание, процессор реагирует на него только тогда, когда приоритет ВУ больше приоритета исполняемой программы. Назначение ВУ различных приоритетов позволяет процессору легко разрешать конфликты, возникающие при одновременном запросе на прерывание от нескольких устройств. Об установке битов приоритета и разрешения прерываний обязан

Рис. 7.4.
Реакция процессора на запрос прерывания от ВУ состоит в «незаметном» для основной (фоновой) программы выполнении программы обработки прерывания.



позаботиться программист. После включения процессора все эти биты автоматически очищаются. Их установка обычно производится специальными командами в самом начале выполнения программы.

Прерывания — не только способ совершенствования механизма обмена данными с ВУ. Очень удобны т. н. *внутренние прерывания*, происходящие при сбоях в работе аппаратуры, неисправностях электропитания, переполнении памяти, недопустимых командах, попытках деления на нуль и т. п. Другие прерывания (программные) инициируются специальными *командами TRAP* («ловушка»). В сложных программных системах «ловушки», вызываемые командами EMT, используются для переключения процессора на выполнение программ, реализующих некоторые стандартные для данной системы функции. Внутренние прерывания имеют свои векторы и обрабатываются аналогично внешним, но в первую очередь.

Обмен данными путем прерываний повышает эффективность работы процессора, исключая непроизводительные циклы ожидания готовности ВУ. Тем не менее определенные потери времени остаются. Действительно, часть времени отнимает загрузка и выгрузка стека, выполнение команды возврата. Более того, подпрограмма обслуживания, использующая РОНы, обязана переслать в ОЗУ их содержимое в момент входа в прерывание и восстановить его перед возвратом, иначе может нарушиться выполнение прерванной программы. Неизбежны «накладные» расходы и на извлечение из ОЗУ самих команд обмена.

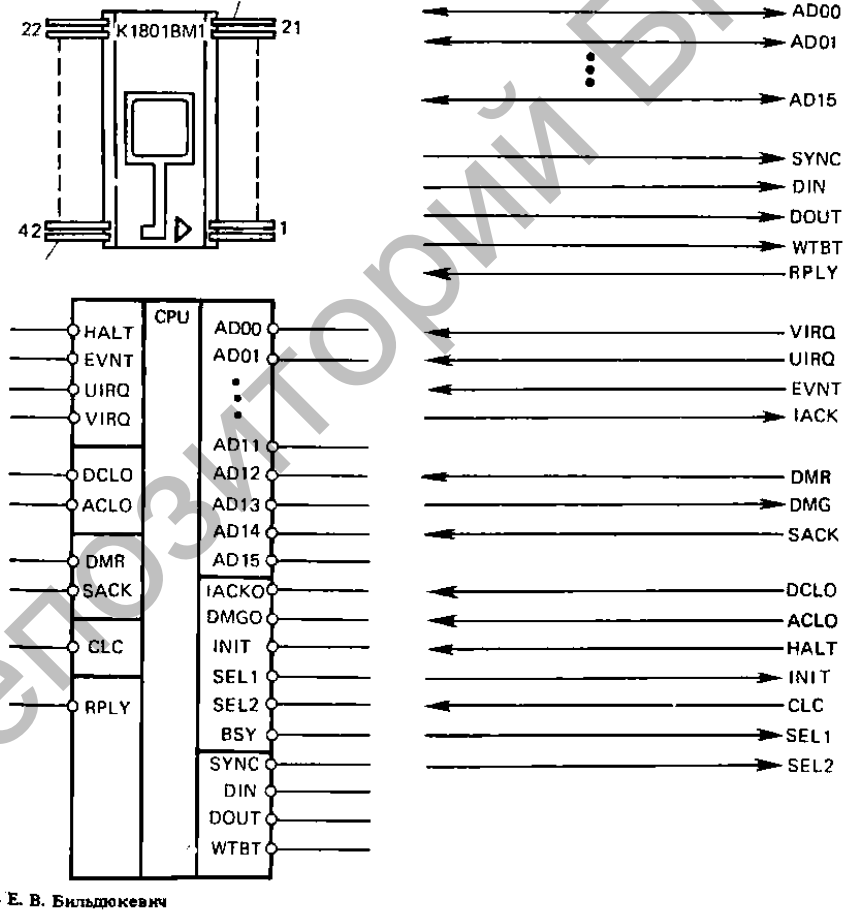
Минимальные потери времени и максимальную скорость обеспечивает обмен данными путем *прямого доступа в память* (ПДП). Получив через особую линию запрос на такого рода обмен, процессор почти полностью отключается от магистрали, позволяя ВУ взаимодействовать непосредственно с ОЗУ. Обмен словом или байтом в этом случае приостанавливает работу процессора на длительность одного цикла ОЗУ. Говорят, что ВУ заимствует или «ворует» этот цикл у процессора, совершенно не изменяя его состояния.

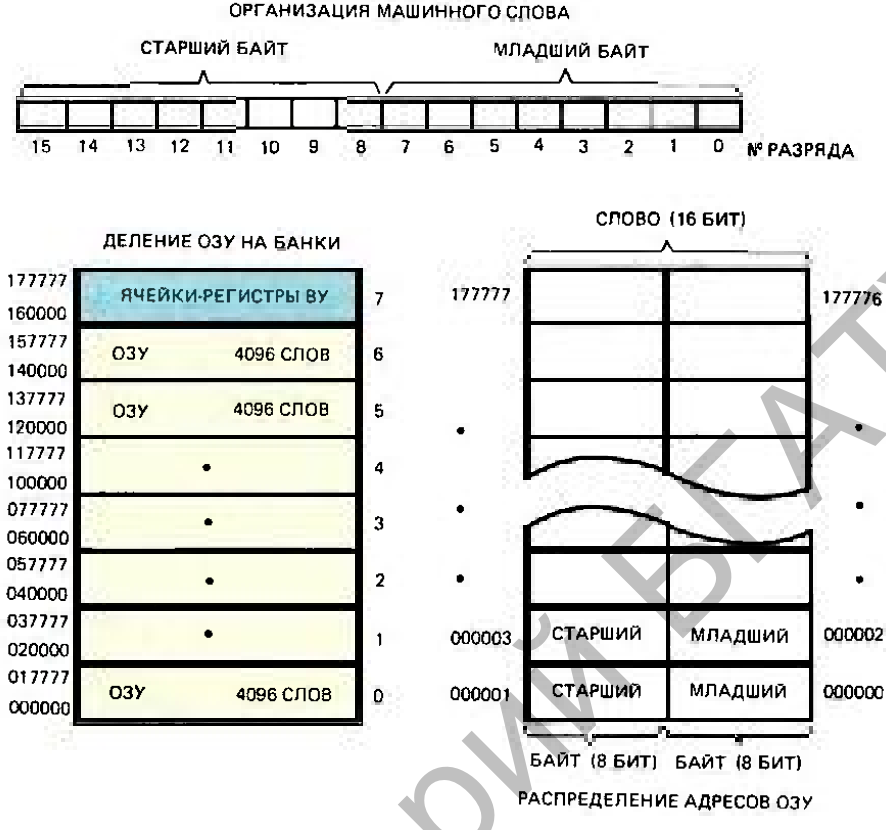
Обмен данными посредством ПДП имеет наивысший приоритет. Если потребность в таком обмене возникает не слишком часто, то процессор, выполняя любую программу, практически не «чувствует» его и оба процесса происходят как бы одновременно. Выигрыш в быстродействии особенно велик при необходимости передачи больших массивов информации, например, при работе с ВЗУ: для самого обмена не нужна программа, а значит, нет потерь времени на выполнение и извлечение команд из ОЗУ. Достоинства ПДП имеют негативную сторону в виде заметного усложнения интерфейса ВУ, полностью берущего на себя управление магистралью.

Линии, образующие магистраль передачи информации, можно разделить на три группы: *информационную, адресную* и *управляющую*. В полном составе все эти шины имеются в магистралях «старших» моделей семейства малых ЭВМ и называются «общая шина». Для магистралей микропроцессоров с целью сокращения числа линий характерно совместное использование одной шины для поочередной передачи вначале адреса, а затем — данных. Именно так построена магистраль микропроцессора К1801ВМ1, носящая название *МПИ* (магистраль передачи информации).

Основу МПИ составляет 16-разрядная *двунаправленная шина*

БИС МП К1801 ВМ1 оформлена в металлокерамическом корпусе, имеющем 42 вывода, которые предназначены для передачи информационных и управляющих сигналов МПИ, а также подключения к источнику питания.





16-разрядные машинные слова позволяют указывать $2^{16} = 65536 =$ = 64К различных адресов. Старшие 8К = 8192 адреса отведены для регистров ВУ, а остальные — для ОЗУ и ПЗУ. Эта часть «адресного пространства» делится на страницы, или банки по 8К адресов, что упрощает создание ЗУ с нужной пропорцией между ОЗУ и ПЗУ (блоки памяти выпускаются с емкостью, кратной 8К адресов).

адреса/данных. Линии управления можно объединить в четыре группы, как это показано на рисунке 7.5.

Каждое ВУ, точнее, любой из его регистров (РД или РС) имеет свой номер — адрес. Этим адресом процессор снабжает любую порцию выводимой информации (слово или байт). Аналогично, каждому слову, вводимому в процессор, предшествует адрес ВУ «отправителя». ЗУ подключаются к МПИ почти так же, как и остальные ВУ, с той лишь разницей, что ячейки памяти имеют другие адреса (рис. 7.6).

Может возникнуть вопрос, почему до сих пор размеры памяти определялись числом адресов, а не слов. Это объясняется тем, что рассматриваемые процессоры способны обращаться не только

к шестнадцатиразрядным словам, но и к каждому из составляющих их байтов. При этом свои адреса имеют каждый из двух байтов любого слова: четные — для младших байтов и нечетные — для старших. Поэтому число слов всегда в два раза меньше числа адресов. Так, выраженные в словах объемы всего адресного пространства и одного банка составляют соответственно 32К и 4К. Для однозначности принято, что адрес слова совпадает с адресом его младшего байта.

Информационное «рукопожатие»

Передача данных между любыми двумя устройствами, подключенными к магистрали, осуществляется по принципу «управляющий — управляемый». Управляющим в каждый момент времени может быть только одно устройство, и именно оно осуществляет все функции по управлению передачей данных по МПИ. Как правило, хозяином МПИ является сам процессор. Перехватить управление магистралью разрешается лишь устройствам прямого доступа в память, таким,как контроллеры накопителей на магнитных лентах, дисках и т. п., и лишь на период обмена с ОЗУ большими порциями данных. В этом случае процессор «освобождает» магистраль, переводя свои выходы в состояние с высоким выходным сопротивлением, и приостанавливает выполнение программы до окончания работы ВУ по пересылке данных.

Сначала рассмотрим случай, когда управляющим устройством является процессор, исполняющий некоторую программу. При исполнении команд он обращается к ячейкам ОЗУ и регистрам внешних устройств с целью:

1) выборки команды из ОЗУ;

2) ввода данных из ОЗУ/считывания содержимого регистров ВУ;

3) записи данных в ОЗУ/регистры ВУ.

В любом случае обмен осуществляется с помощью одного или нескольких *циклов* магистрали, за каждый из которых по шине адреса/данных передается одно машинное слово или байт данных. Каждый цикл имеет свой *протокол*, т. е. порядок обмена сигналами между ведущим и ведомым устройствами. В МПИ реализуются циклы 4 видов:

1) ввод данных;

2) вывод данных (вывод слова);

3) вывод байта;

4) ввод-пауза-вывод.

Общее для всех циклов — *асинхронный характер*. В данном случае это означает, что на каждый управляющий сигнал должен поступить сигнал ответа от управляемого устройства (обмен «рукопожатием»), причем время ожидания сигнала ответа (таймаут) ограничено величиной 10 мкс. Если в течение таймаута управляющее устройство не получило ответа, происходит внутреннее прерывание.

Асинхронное выполнение циклов передачи данных позволяет вести с максимально высоким темпом обмен информацией между устройствами, обладающими самым различным быстродействием.

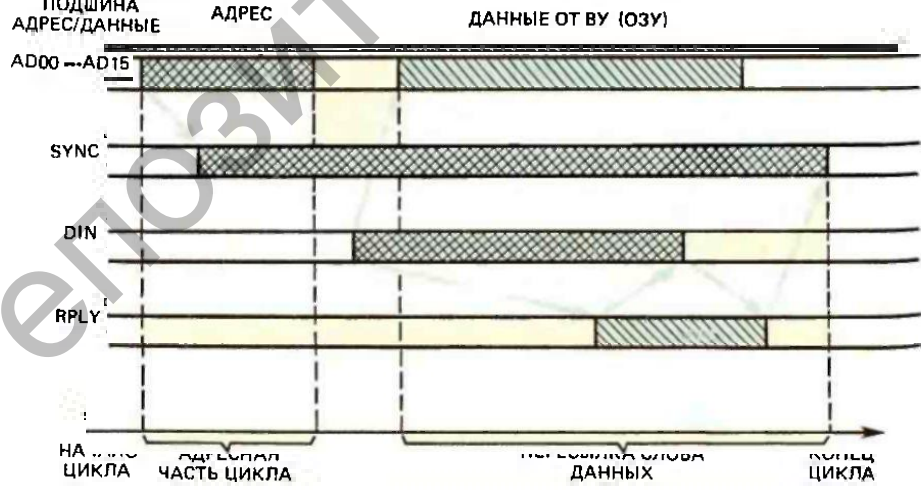
Циклы магистрали

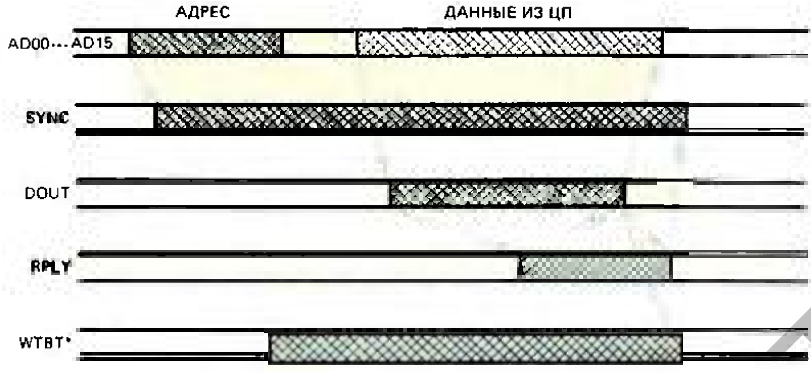
Каждый из 4 циклов МПИ начинается с адресации ячейки ОЗУ или регистра ВУ, к которым будет производиться обращение. В ходе *адресной части цикла* процессор устанавливает нужный адрес на шине адреса-данных (линии AD00-AD15) и вырабатывает **сигнал синхронизации адреса SYNC** (SYNCronization), который действует в течение всего цикла. С появлением сигнала SYNC переданный адрес воспринимается всеми управляемыми устройствами, подключенными к МПИ, и адресная часть цикла заканчивается. За адресной частью цикла следует информационная, в которой выполняется конкретная операция: ввод, вывод, вывод байта, ввод-пауза-вывод. В информационной части цикла принимает участие лишь то управляемое устройство, которое имеет в своем составе ячейку (регистр) с адресом, принятым в адресной части цикла, что обеспечивается так называемым селектором (де-

После адресной части цикла «ввод» ЦП подает сигнал DIN (Data IN — ввод данных). В ответ на это ВУ устанавливает требуемые данные на линиях AD00-AD15 и формирует сигнал ответа RPLY (RePLY — отзыв). Получив сигнал RPLY, процессор принимает данные и снимает сигнал DIN. Вслед за этим устройство снимает сигнал RPLY, а процессор — сигнал SYNC. На этом и следующих рисунках используются такие обозначения:

— сигнал передается управляющим устройством (процессором);

— сигнал передается управляемым устройством



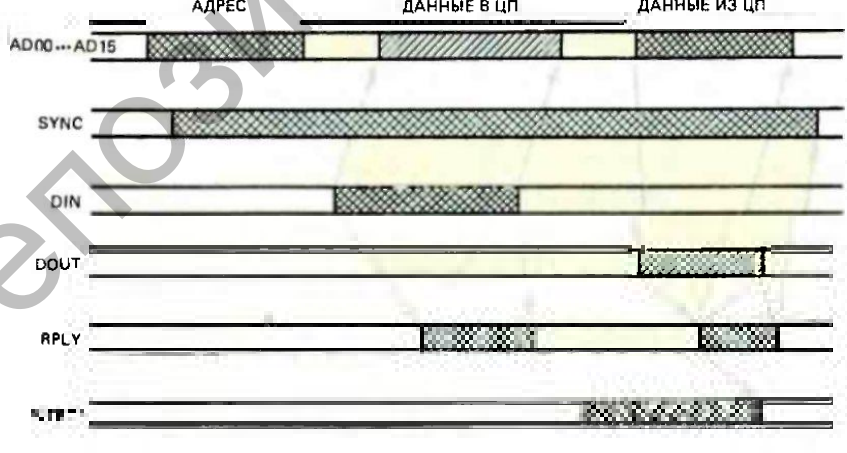


После адресной части цикла «вывод» ЦП устанавливает на шине адреса/данных выводимое слово (или байт), а затем формирует сигнал DOUT (Data OUT – вывод данных). Выбранное ВУ, получив сигнал DOUT, принимает данные с шины адреса/данных, после чего формирует сигнал ответа RPLY. Получив ответ, процессор снимает сигналы DOUT, SYNC и данные с линий AD00-AD15. WTBT устанавливается только в цикле «вывод байта».

шифратором) адреса. Назовем такое устройство *выбранным*.

Цикл «ввод». В течение *цикла «ввод»* (рис. 7.7) производится передача машинного слова из ОЗУ или регистра ВУ в ЦП. При исполнении любой команды ЦП по крайней мере один раз выполняет цикл «ввод» с целью ее выборки.

Цикл «ввод-пауза-вывод» напоминает цикл «ввод», но по окончании приема данных сигнал SYNC не снимается, а сразу следует информационная часть цикла «вывод». Адрес, принятый в адресной части цикла, сохраняется неизменным в течение всего времени действия сигнала SYNC. WTBT устанавливается при операциях с байтами.



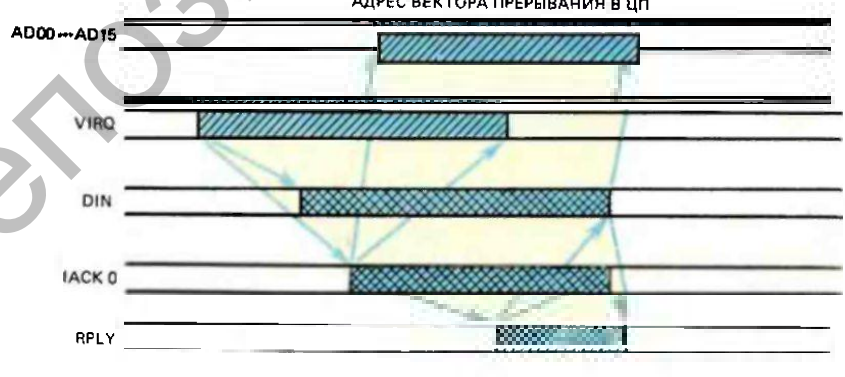
Циклы «вывод» и «вывод байта». При исполнении *циклов «вывод»* и *«вывод байта»* производится передача из процессора в ячейку ОЗУ или регистр ВУ соответственно одного машинного слова или байта. В цикле «вывод байта» передается лишь 1 байт данных. При этом в информационной части цикла устанавливается **сигнал WTBT** (WriTe ByTe — запись байта), как это показано на рисунке 7.8.

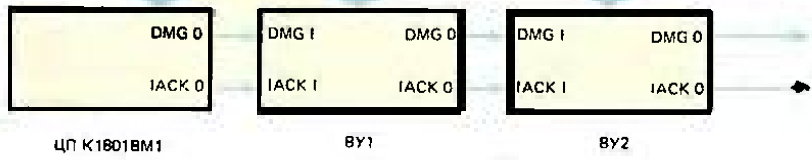
Цикл «ввод-пауза-вывод». При исполнении многих команд (например, увеличения на 1) возникает необходимость в модификации содержимого выбранной ячейки ОЗУ. Эту операцию можно произвести с помощью последовательного исполнения цикла «ввод» и цикла «вывод», но поскольку адрес ячейки не меняется, то можно обойтись без его повторной передачи, а два отдельных цикла объединить в один. Такой цикл называется *циклом «ввод-пауза-вывод»* (рис. 7.9).

Прерывание и прямой доступ в память

Устройство, которому требуется прервать программу, вырабатывает **сигнал VIRQ** (Vector Interrupt ReQuest — запрос векторного прерывания). Процессор, получив запрос VIRQ, удовлетворяет требование путем выдачи **сигналов DIN** и **IACK O** (Interrupt ACKnowledge Output — выходной сигнал предоставления прерывания). Получив сигналы DIN и IACK O, прерывающее устройство снимает сигнал VIRQ и устанавливает на линиях AD00-AD15 адрес «своего» вектора прерывания, сопровождая его **сигналом ответа RPLY** (как и в цикле «ввод»). Приняв адрес вектора прерывания, ЦП снимает сигналы DIN и IACK O и переходит к подпрограмме обслуживания прерывания, а устройство снимает сигнал RPLY. На этом процесс предоставления прерывания заканчивается. Временная диаграмма этой процедуры представлена на рисунке 7.10.

Процедура прерывания программы происходит по инициативе ВУ и предусматривает передачу адреса вектора прерывания в ЦП.

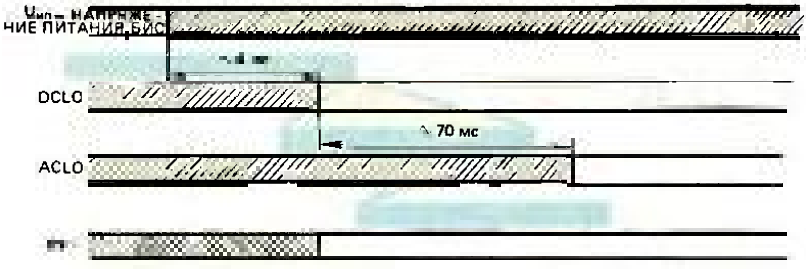




В отличие от остальных сигналов МПИ, сигналы предоставления прерывания и прямого доступа в память проходят «транзитом» через все ВУ. Обозначение входного для ВУ сигнала предоставления принято оканчивать буквой I (Input), а выходного — О (Output). Поэтому одна и та же линия, соединяющая два устройства, именуется по-разному с каждого конца.

В МПИ имеются специальные линии для организации двух так называемых *радиальных прерываний*, не требующих передачи адреса вектора прерывания. Эти линии носят название **UIRQ** (User Interrupt ReQuest — запрос пользовательского прерывания) и **EVNT** (EVeNT — событие, для прерывания от таймера микропроцессорной системы). При подаче сигналов на эти линии процессор автоматически выполняет переход на подпрограммы обработки прерываний с адресами векторов соответственно 270_8 и 100_8. Сигналы IACK O и DIN при этом не вырабатываются, а сигнала ответа RPLY не требуется. Линия EVNT обычно возбуждается с частотой 50 Гц, что позволяет специальной программе вести учет текущего времени («часы»). Линия UIRQ предназначена для использования разработчиком микропроцессорной системы по своему усмотрению.

Отметим одну важную особенность в организации прерываний. Прерывание программы, как и предоставление прямого доступа в память, происходит по инициативе внешнего устройства. В системе с множеством внешних устройств возникает опасность того, что в один и тот же момент прервать программу понадобится не одному, а сразу нескольким ВУ. По этой причине в МПИ предусмотрена защита от ошибочных ситуаций, связанных с одновременной передачей сразу нескольких векторов прерывания или «захватом» магистрали сразу несколькими устройствами прямого доступа в память. Она заключается в том, что линии, несущие сигналы IACK O и DMG O (выходные сигналы предоставления прерывания и прямого доступа в память), проходят последовательно через все устройства (рис. 7.11). Любое устройство, получив в ответ на требование прерывания или прямого доступа в память соответствующий сигнал предоставления, обязано прежде всего запретить его дальнейшее распространение, т. е. прохождение к следующему устройству, а затем реализовать операции прямого доступа или прерывание программы. Если же оно не требовало прерывания или прямого доступа в память, то соответствующие



С целью прямого доступа в память ВУ вырабатывает сигнал требования ПДП DMR (Direct Memory access Request). Завершив текущую команду, процессор приостанавливает выполнение программы и формирует сигнал DMG O (Direct Memory access Grant — предоставление ПДП). Приняв сигнал DMG O, ВУ запрещает его дальнейшее прохождение и формирует сигнал SACK (Selection ACKnowledge — подтверждение выбора). В ответ процессор снимает сигнал DMG O и «замирает» на период действия сигнала SACK, т. е. до окончания обмена.

сигналы предоставления должны быть переданы на следующее устройство.

Вам приходилось, наверное, видеть, как в час «пик» полупустой трамвай за несколько секунд заполняется пассажирами. Большинство из них сразу же пытается передать «пробить» проездные талоны. В суматохе уже не так просто разобраться, кому какой талон принадлежит. Чаще всего поступают так: кто стоит ближе к компостеру, получает первый попавшийся, а остальные передает дальше по цепочке. Что-то похожее имеет место при приоритетной организации процедур предоставления прерываний и прямого доступа в память. Таким образом устройство, расположенное на магистрали ближе к процессору, обладает и большим приоритетом на обмен.

Процедура предоставления *прямого доступа в память* (ПДП) связана с полным освобождением процессором магистрали на период пересылки данных между ОЗУ и ВУ (рис. 7.12). Управление МПИ при этом берет на себя само внешнее устройство. После окончания пересылок ВУ освобождает МПИ и процессор продолжает выполнение прерванной программы, для которой сеанс прямого доступа в память проходит почти «незаметно».

Начальный пуск и синхронизация микропроцессора

МПИ содержит несколько вспомогательных линий, осуществляющих управление запуском микропроцессора после включения питания. К ним относятся **линии ACLO** (Alternative Current LOw — авария сетевого питания), **DCLO** (Direct Current LOw — авария источника питания), **INIT** (INITialization — начальная установка магистрали), **SEL1** и **SEL2**. Сигналы ACLO и DCLO вырабатываются источником питания микропроцес-

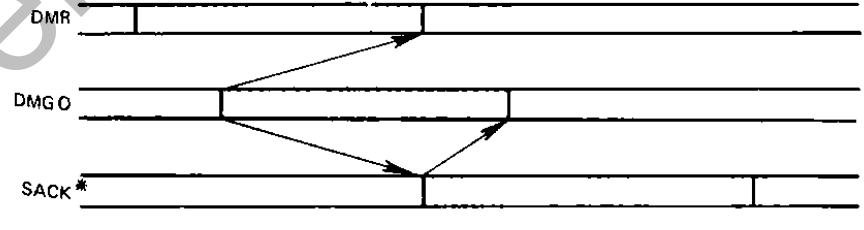
сорной системы и являются входными для ЦП (рис. 7.13).

Сигнал INIT подается процессором при выполнении специальной команды «сброс», запуске программы, в момент включения питания и служит для установки всех устройств на магистрали в исходное состояние.

Начальный пуск микропроцессора представляет собой переход к исполнению некоторой программы с определенного адреса в момент окончания действия сигнала ACLO. Адрес начального пуска микропроцессора задается с помощью специального внешнего *регистра начального пуска* (РНП), имеющего адрес 177716_8. По окончании действия сигнала ACLO микропроцессор автоматически осуществляет чтение содержимого РНП, заносит его в счетчик команд и приступает к выполнению программы со сформированного таким образом адреса. Для упрощения устройства регистра начального пуска микропроцессор имеет специальный вывод SEL1, на котором появляется сигнал в момент чтения РНП. По этому сигналу схема, реализующая РНП, должна выставить на линии AD00-AD15 адрес начального пуска, а подача сигнала RPLY при этом не требуется. Отметим, что с помощью РНП можно задавать адрес начального пуска, кратный 400_8 (младший байт СК очищается).

Обычно в качестве адреса начального пуска используют *стартовый адрес* программы начального загрузчика операционной системы с магнитного диска, записанной в постоянном запоминающем устройстве, либо (при использовании МП в качестве контроллера) стартовый адрес управляющей программы, также хранящейся в ПЗУ. Во многих микро-ЭВМ используется специальное ПЗУ, содержащее *программу-эмулятор пульта*, которой может передаваться управление в момент начального пуска. Эта программа позволяет использовать терминал микро-ЭВМ в качестве пульта управления, который обычно имеется у мини- и больших ЭВМ. Программа-эмулятор принимает с терминала и выполняет некоторый набор команд типа «распечатать содержимое ячейки ОЗУ», «начать выполнение программы с определенного адреса» и т. п. (подробнее об этом см. 8.7). Еще одна особенность МП К1801ВМ1 и большинства других — необходимость в так называемом «так-

Для осуществления нормального запуска МП К1801ВМ1 сигналы ACLO и DCLO должны подаваться в указанном на временной диаграмме порядке при включении питания и в обратном порядке при выключении.



товом питании», которое представляет собой синхросигнал, вырабатываемый отдельным генератором и подаваемый на специальный вход синхронизации микропроцессора CLK. Он осуществляет синхронизацию работы внутренних узлов МП, т. е. является «жизненно важным» для нормальной работы микропроцессорных БИС.

Радиальные интерфейсы

Очень большая, но конечная, скорость распространения электрических сигналов по проводам заставляет снижать темп обмена данными через МПИ с увеличением ее длины. Как следствие все входящие в систему ВУ должны размещаться в непосредственной близости от ЦП, что не всегда удобно. Поэтому для подключения удаленных устройств часто используют *контроллеры радиальных интерфейсов*. К каждому из таких контроллеров можно подключить лишь одно ВУ, соединив его с контроллером более длинной низкоскоростной линией связи, которая состоит из относительно небольшого числа проводов. Простейший *радиальный интерфейс ИРПР* (интерфейс радиальный параллельный) позволяет передавать данные в параллельном виде на расстояние до 15...20 м с помощью следующих сигналов:

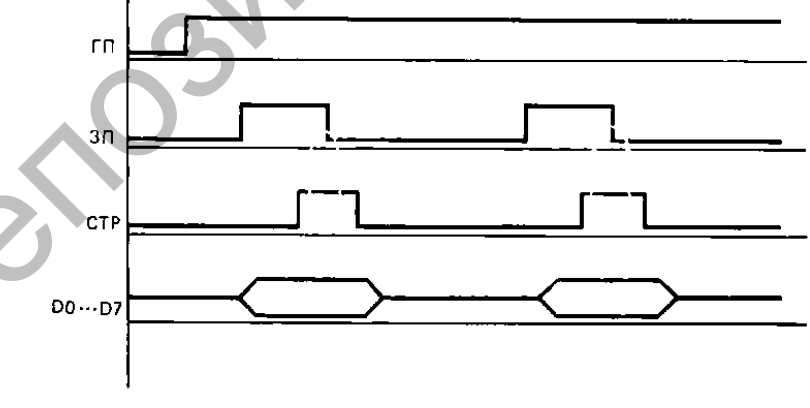
ГП *(готовность приемника)* — формируется приемником информации и означает, что приемник работоспособен и подключен;

ЗП *(запрос приемника)* — означает, что приемник готов к приему очередного байта данных;

СТР *(строб источника)* — поступает от источника и информирует приемник о том, что на линии данных установлен очередной байт;

D0...D7 — данные.

В интерфейсе ИРПР прием данных приемником осуществляется по сигналу СТР. Признаком получения данных служит снятие сигнала ЗП.



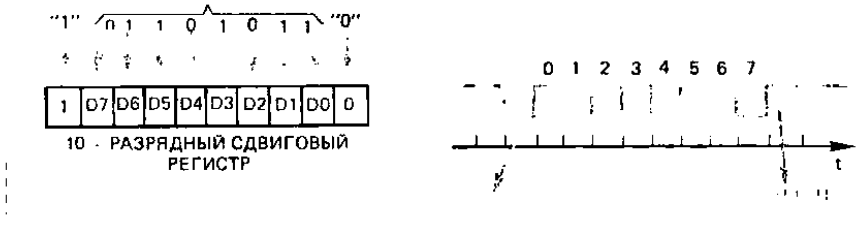


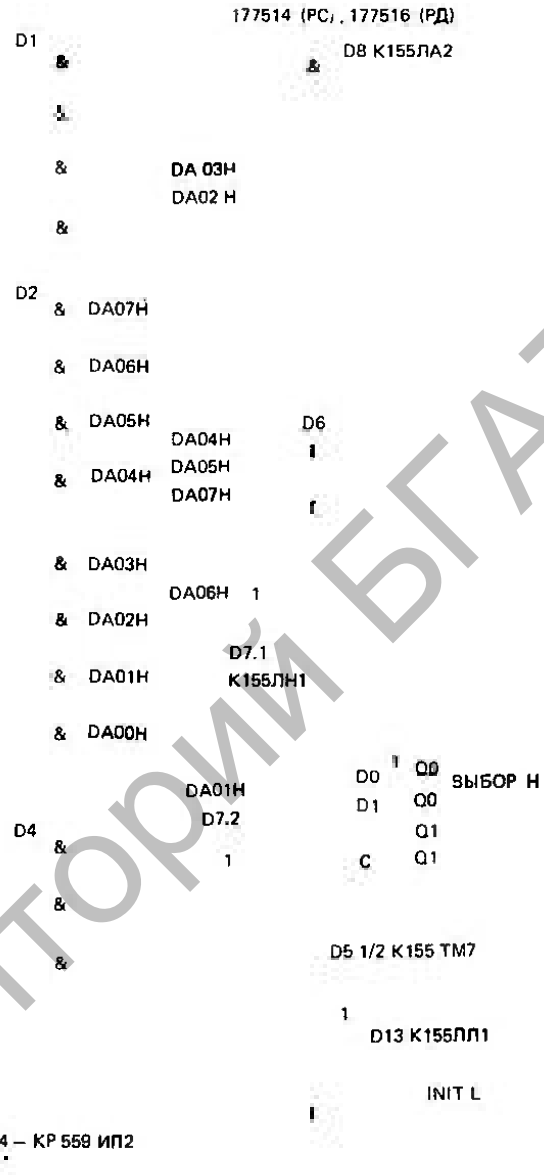
Рис. 7.13

При обмене посредством ИРПС байт данных снабжается стартовым, одним или двумя стоповыми битами и высылается в линию связи последовательно, т. е. каждый бит передается в течение строго фиксированного промежутка времени, называемого битовым интервалом. Единица скорости передачи данных — бод, т. е. бит в секунду. На практике используют следующие стандартные скорости передачи: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19 200 бод.

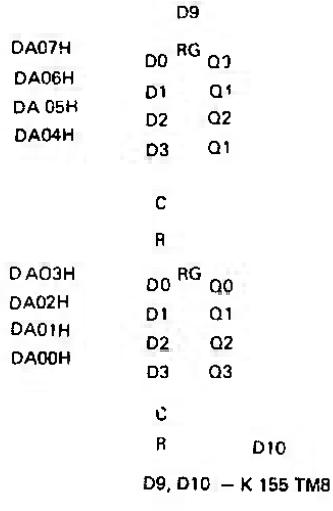
Протокол обмена посредством ИРПР поясняется рисунком 7.14. Для передачи данных в обоих направлениях количество линий удваивается.

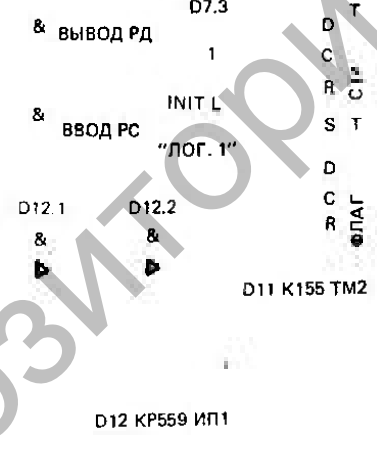
Пример конкретной реализации ИРПР для желающих уяснить устройство интерфейсов представлен на рисунке 7.16. Со всеми элементами этой схемы читатель уже знаком. Исключение — специальные приемники и передатчики для обмена сигналами с МПИ. Приемники обладают низкими входными токами и высокой помехоустойчивостью. Передатчики имеют мощный выходной каскад и, подобно тристабильным элементам, допускают объединение своих выходов для реализации *функции «монтажное ИЛИ»*. Работа интерфейса отвечает описанным выше требованиям протоколов МПИ и ИРПР. Неиспользуемые разряды РС и все разряды РД в цикле «ввод» считываются в процессор нулями.

Относительно большое число линий является серьезным недостатком интерфейса ИРПР. Более удобен в этом отношении интерфейс ИРПС, или «токовая петля», применяемый для передачи данных между вычислительной системой и удаленными на расстояние до нескольких километров видеотерминалами, печатающими устройствами. В интерфейсе ИРПС передача данных в каждом направлении осуществляется по линии связи, состоящей из двух скрученных проводов (витая пара). Для передачи логической единицы источник данных посылает в линию ток величиной 20 или 60 мА в течение фиксированного промежутка времени, называемого битовым интервалом, а при передаче логического нуля на время битового интервала ток выключается. В зарубежной аппаратуре при передаче данных вместо импульсов тока зачастую используют уровни напряжения: единица представляется положительным напряжением от 3 до 12 В, нуль — отрицательным напряжением



Упрощенная схема байтового ИРПР для устройства печати работает в режиме программного обмена. Интерфейс имеет два программно-доступных регистра — РС с адресом 177514 и РД с адресом 177516. РС доступен процессору только для счи-

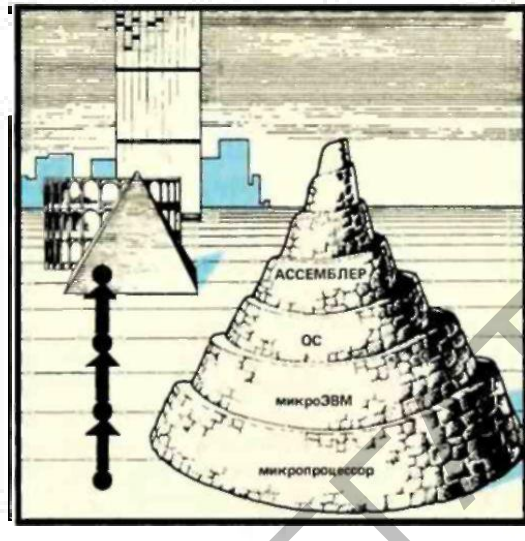




тывания, а РД — для записи. Сигналы с суффиксом L передаются активным низким уровнем, т. е. напряжение 0...0,5 В означает наличие сигнала, а 2,4...5 В — отсутствие (логический «0»). Сигналы с префиксом В относятся к МПИ.

той же величины. Перед передачей с помощью сдвиговых регистров производится преобразование данных из параллельной формы в последовательную, а в точке приема — обратное преобразование (рис. 7.15). В процессе преобразования в последовательный код к данным добавляют специальные служебные биты — стартбит (логический «0») и один или более стоп-битов (логическая «1»), которые используются приемником для определения моментов начала и конца передачи байта. В отсутствие передачи данных линия находится в состоянии логической единицы. Все необходимые для последовательной передачи преобразования данных выполняются с помощью асинхронных приемопередатчиков, реализованных обычно в виде БИС.

Более сложные преобразования данных необходимы при передаче цифровой информации по обычной телефонной сети, поскольку в этом случае передача и прием ведутся по одной и той же паре проводов. Кроме того, телефонный канал пропускает сигналы в узком диапазоне частот — от 300 до 3000 Гц. Поэтому для передачи цифровой последовательности применяют специальный адаптер телефонной линии — модем (сокращенно от модулятор — демодулятор), в котором при передаче логические ноль и единица представляются тональными посылками двух различных частот (например, 1080 и 1750 Гц), а принимаемый с линии сигнал пропускается через фильтры и, таким образом, демодулируется. В силу частотных ограничений темп передачи данных по телефонным линиям связи обычно не превышает 600 бит/с.



ОТ МИКРОПРОЦЕССОРА К ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЕ

Процессор, в том числе МП, и остальное вычислительное оборудование (ЗУ, ВУ — все то, что на языке программистов именуется *hardware* — «жесткая компонента» или попросту «железо») сами по себе беспомощны. «Вдохнуть жизнь» в аппаратуру позволяет *программное обеспечение* (ПО, *software* — «мягкая компонента»): будь то несложная программа в ПЗУ микропроцессорного контроллера или целый комплекс программ для высокопроизводительной ЭВМ, одновременно обслуживающей множество терминалов. В последнем случае ПО настолько преображает возможности самой ЭВМ, что программист, работающий за терминалом, имеет дело как бы с другой, «виртуальной» машиной, а точнее — с *вычислительной системой*, понимающей слова и даже отдельные предложения на языке, близком к естественному.

Реально же ни МП, ни ЭВМ не умеют выполнять ничего, кроме ограниченного набора достаточно простых команд, представленных в ОЗУ или ПЗУ комбинациями нулей и единиц. Удобная для человека форма диалога — результат осуществляемой специальными программами интерпретации команд «виртуальной» машины. Даже для одного типа ЭВМ или МП можно создать множество новых языков диалога. В зависимости от степени близости к языку машины, их делят на *языки низкого* и *высокого уровня*. В этой главе будет рассмотрен язык самого низкого, исходного уровня для МП К1801 и сходных с ним ЭВМ — язык машинных команд. Именно на этот уровень, как на фундамент, надстраиваются «этажи» ПО, обеспечивающие более комфортный для программиста характер общения с вычислительной техникой.

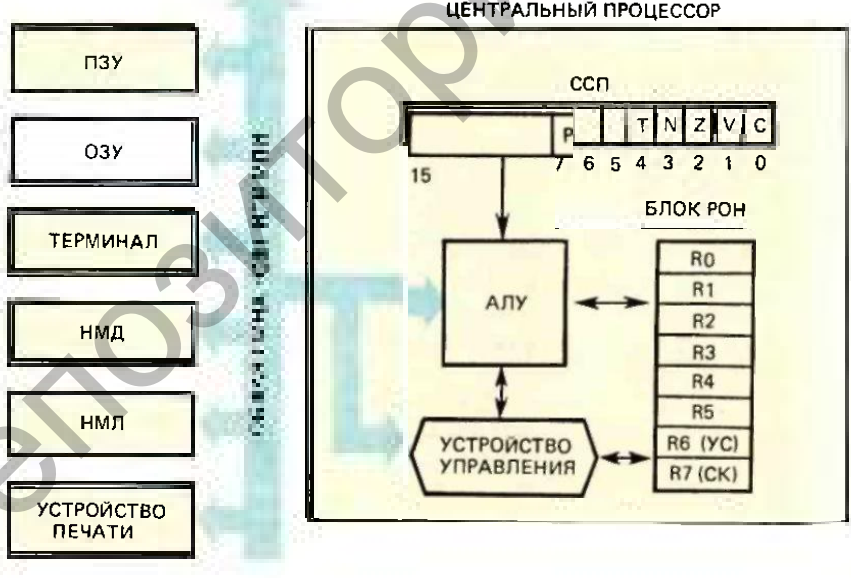
У читателя все эти знания уже есть. Напомним о них.

С точки зрения программиста ЦП представляет собой совокупность узлов, изображенных на рисунке 8.1. УУ координирует работу всех устройств. Каждая команда программы считывается с ОЗУ или ПЗУ и декодируется *(фаза выборки команды)*, затем при необходимости вычисляются адреса операндов, которые извлекаются из памяти *(фаза операндов)*, после чего выполняется заданное действие *(фаза исполнения)*. Сложность фазы операндов зависит от используемых способов адресации.

Большинство действий над операндами производится в АЛУ. Сложные операции выполняются путем многократного выполнения базовых операций АЛУ. Заметим, что АЛУ не осуществляет операций по пересылке данных, а лишь оперирует данными, поданными заблаговременно на его входы. Поэтому его нередко сравнивают со слепым жонглером, который потрясает зрителей своими головокружительными трюками с горящими факелами, но лишь после того, как их вручат ему в руки.

Для хранения слова состояния процессора служит специальный регистр, в отдельные биты которого УУ заносит информацию об особых случаях при выполнении некоторых действий. *Z-бит* и

В состав центрального процессора входят АЛУ, УУ, блок РОН и регистр ССП. Два последних узла доступны программисту: их состояния могут изменяться при выполнении команд.



N-бит устанавливаются, соответственно, при нулевом и отрицательном результатах (для представления отрицательных чисел используются дополнительные коды). *C-* и *V-биты* сигнализируют о переполнении при работе с натуральными и дополнительными кодами. Единичное значение *бита P* ССП запрещает внешним устройствам прерывания программы. Не рассматривавшийся до сих пор *бит T* (Trace — прослеживание) используется при отладке программ. Его единичное состояние вызывает внутреннее прерывание с адресом вектора 14_8 после исполнения каждой команды, что удобно, например, для выполнения программы по шагам при ее отладке.

В ЦП имеется 8 регистров общего назначения R0...R7. Регистры R6 и R7 наделены особыми функциями. Регистр R7 играет роль счетчика команд и всегда содержит адрес очередной подлежащей исполнению команды.

В процессе выборки команды УУ устанавливает на магистрали содержимое R7 и производит цикл «ввод». Слово, записанное по этому адресу в ОЗУ или ПЗУ, интерпретируется как очередная команда. Сразу после выборки команды (до ее исполнения) содержимое R7 автоматически увеличивается на 2, т. е. указывает адрес следующей команды.

R6 служит *указателем системного стека* (УС), в качестве которого может быть использована произвольная область ОЗУ (обычно — в «нижней» части памяти). УС всегда содержит адрес последней занятой ячейки, а в начале работы — верхнюю границу стека. К УС разрешено обращаться с помощью команд для работы с подпрограммами, некоторых команд, использующих R6, или как к одному из РОН. Обычно значение R6 уменьшается на 2 перед записью каждого слова в стек и увеличивается на 2 после каждого считывания. Такой механизм обеспечивает принцип «первым вошел, последним вышел», хотя сами элементы (слова) не перемещаются в ОЗУ. Изменяется лишь нижняя граница занимаемой стеком области ОЗУ. Аппаратный доступ к стеку используется в процессах обработки прерываний для хранения значений СК и ССП прерываемой программы.

Напомним, что ЦП рассматривает подключаемую к нему память как совокупность 64 Кбайт (32 Кслов) с адресами $0...177777_8$, причем адреса 0...376 обычно зарезервированы для векторов прерываний, а старшие 8 К адресов — для регистров ВУ. Это позволяет обойтись без специальных команд ввода-вывода, т. к. к регистрам ВУ ЦП может обращаться так же, как и к ОЗУ.

Если по исполняемым функциям все команды ЦП можно разделить на команды пересылки, управления и арифметико-логические (см. 2.11), то с точки зрения кодирования удобнее классификация по форматам (см. 3.7).

8.2. **Методы адресации**

Методы адресации весьма важны для понимания последующего материала.

Ограничимся рассмотрением единственной **команды очистки** содержимого регистра или ячейки памяти. Она относится к числу одноадресных (см. рис. 3.9), и ее восьмеричный код можно условно записать в виде $0050DD_8$ (либо $1050DD_8$ — для байтов), где $^0/_1050$ — код операции, DD — восьмеричные цифры, выражающие, соответственно, номера метода адресации и используемого при этом РОН. Наряду с восьмеричной записью используется и символическая запись, характерная для обозначения команд в языке *Ассемблер*. В этой форме записи КОП представляется в виде мнемонического английского обозначения, а остальные особенности станут ясными из примеров, иллюстрирующих механизм каждого из 8 основных способов адресации. В этих примерах через косую черту приводятся значения адреса и содержимого ячеек памяти (регистров), участвующих в процессе выполнения команды. Для самой команды очистки, имеющей мнемоническое обозначение **CLR** (от CLeaR — очистить), отведем постоянное место в ОЗУ по адресу 001000_8.

Метод 0. Регистровая адресация. Предполагается, что в указанном командой регистре находится сам операнд.

Пример: CLR R3

до выполнения	после выполнения
R3/123574	R3/000000
001000/005003	001000/005003

Метод 1. Косвенно-регистровая адресация. В указанном РОН содержится не сам операнд, а его адрес в ОЗУ.

Пример: CLR (R5)

до выполнения	после выполнения
R5/001216	R5/001216
001000/005015	001000/005015
001216/165123	001216/000000

Этот метод обеспечивает доступ к любой ячейке или байту ОЗУ.

Метод 2. Автоинкрементная адресация. Как и в методе 1, операнд находится по адресу, содержащемуся в указанном РОН, но после выполнения команды содержимое РОН автоматически увеличивается на 2 (или 1, если команда оперирует с байтами), указывая тем самым на следующее слово (байт) в ОЗУ.

Пример 1: CLR (R5)+

до выполнения	после выполнения
R5/020000	R5/020002
001000/005025	001000/005025
020000/145612	020000/000000

Пример 2: CLRB (R5)+

до выполнения	после выполнения
R5/020000	R5/020001
001000/105025	001000/105025
020000/145612	020000/145400

Метод 3. Косвенно-автоинкрементная адресация. Указанный в команде РОН хранит адрес ячейки, содержащей адрес операнда. После выполнения операции содержимое РОН увеличивается на 2.

Пример: CLR @ (R5)+*

до выполнения	после выполнения
R5/002066	R5/002070
001000/005035	001000/005035
002066/010100	002066/010100
010100/053530	010100/000000

Метод 4. Автодекрементная адресация. Содержимое РОН вначале уменьшается на 2 (или на 1 в случае байтовой команды), а затем используется в качестве адреса операнда. Обратите внимание на отличие в мнемонических обозначениях от автоинкрементного метода.

Пример: CLR —(R0)

до выполнения	после выполнения
R0/007000	R0/006776
001000/005040	001000/005040
006776/123456	006776/000000
007000/025340	007000/025340

Основное назначение методов 4 и 2 — обработка данных, расположенных в последовательных ячейках памяти.

Метод 5. Косвенно-автодекрементная адресация. Содержимое РОН уменьшается на 2, а затем используется в качестве адреса ячейки, содержащей адрес операнда.

Пример: CLR @—(R1)

до выполнения	после выполнения
R1/102520	R1/102516
001000/005051	001000/005051
102516/123560	102516/123560
123560/105407	123560/000000

Методы 5 и 3 позволяют организовать в ОЗУ «программные» стеки, использующие в качестве указателя любой из РОН.

* Значок @ (читается «эт») — еще одно, наряду с круглыми скобками, обозначение косвенной адресации.

Следующие 2 метода адресации на первый взгляд «запутанные», но тем не менее быстро запоминаются и помогают писать эффективные и быстро выполняющиеся программы.

Наличие в команде операнда, адресуемого методом 6 или 7, вызывает автоматическое извлечение слова, следующего непосредственно за командой, которое используется при вычислении адреса и называется индексным словом. Таким образом, команда с методом адресации 6 или 7 всегда занимает 2 последовательные ячейки памяти, т. к. записывается вместе с некоторым индексным словом, или же 3 ячейки, когда команда двухоперандная и оба операнда адресуются по методам 6 или 7. Для того чтобы индексные слова не воспринимались как команды, что приводило бы к ошибкам, процессор автоматически производит коррекцию СК.

Метод 6. Индексная адресация. Адрес операнда определяется как сумма индексного слова и содержимого указанного регистра общего назначения.

Пример: CLR 13004(R2)

до выполнения	после выполнения
R2/100512	R2/100512
001000/005062	001000/005062
001002/013004	001002/013004
113516/053720	113516/000000

Метод 7. Косвенно-индексная адресация. Сумма содержимого, указанного РОН и индексного слова, определяет адрес ячейки, содержащей адрес операнда.

Пример: CLR @ 13004(R2)

до выполнения	после выполнения
R2/100512	R2/100512
001000/005072	001000/005072
001002/013004	001002/013004
113516/053720	113516/053720
053720/044510	053720/000000

Все восемь методов адресации могут быть использованы в любых одно- и двухоперандных командах. В последнем случае каждому из операндов может соответствовать свой метод адресации.

8.3. Использование счетчика команд для адресации операндов

СК, т. е. РОН R7, может использоваться в командах со всеми рассмотренными методами адресации, однако в таком сочетании только четыре из них имеют практическое значение.

Помимо команды CLR, в примерах адресации с применением СК используем **команду сложения** (мнемоническое обозначение **ADD**), относящуюся к классу двухадресных (рис. 3.6). Поскольк-

ку код операции сложения — 06_8, то ее восьмеричная запись имеет вид

06SSDD,

где SS, DD — восьмеричные номера метода адресации и используемого регистра для операндов источника (S) и приемника (D). Вследствие достоинств, вытекающих из особой функции R7, эти 4 метода получили специальные названия и мнемонические обозначения:

1. Непосредственная адресация (СК + метод 2). Операнд выбирается из ячейки, следующей за командным словом.

Пример: ADD #2, R0

до выполнения	после выполнения
R0/004436	R0/004440
001000/062700	001000/062700
001002/000002	001002/000002

2. Абсолютная адресация (СК + метод 3). Из ячейки, следующей за командным словом, выбирается адрес операнда.

Пример: ADD @ #200, R0

до выполнения	после выполнения
R0/004430	R0/012070
001000/063700	001000/063700
001002/000200	001002/000200
000200/005440	000200/005440

3. Относительная адресация (СК + метод 6). Адрес операнда определяется как сумма содержимого СК и индексного слова. Заметим, что в момент суммирования с индексным словом СК скорректирован, т. е. содержит адрес очередной команды.

Пример: CLR 1120

до выполнения	после выполнения
001000/005067	001000/005067
001002/000114	001002/000114
001004/	001004/
001120/054100	001120/000000

4. Косвенно-относительная адресация (СК + + метод 7). Адрес операнда выбирается из ячейки, адрес которой равен сумме содержимого СК и индексного слова.

Пример: CLR @ 1120

до выполнения	после выполнения
001000/005077	001000/005077
001002/000114	001002/000114
001004/	001004/
001120/054100	001120/054100
054100/021577	054100/000000

Режимы адресации с использованием СК позволяют легко составлять программы, которые не требуют никакой переделки при необходимости изменить их положение в ОЗУ *(позиционно-независимый код)*, а также эффективно работать с данными, не организованными в массивы.

Особо отметим, что непосредственный и абсолютный методы адресации позволяют записывать прямо в тексте программы фиксированные адреса и константы. Фактическая длина команды в этом случае может составлять 2 или 3 машинных слова.

Безусловный переход. Подпрограммы и стеки

Любую сложную программу удобно разбить на отдельные части (фрагменты), которые можно независимо друг от друга разрабатывать и отлаживать. В этом случае собственно программа может состоять в основном из команд перехода к своим фрагментам, которые после выполнения должны передавать управление назад в основную программу. Однако при таком подходе затруднено многократное использование одних и тех же частей, например, при различных значениях аргументов *(параметров)*.

Мощный метод повышения эффективности — использование *механизма подпрограмм*, позволяющего избежать дублирования одинаковых фрагментов, а также легко включать в программы заранее написанные модули из библиотеки готовых программ. Как и для одноадресной (см. рис. 3.9) **команды JMP** (JuMP — прыгать) безусловной передачи управления, имеющей код 0001DD, обращение к подпрограмме приводит к выполнению группы команд, начиная с указанного адреса. Отличие состоит в том, что с целью обеспечить возврат к нужному месту основной программы, процессор обязан запоминать текущее содержимое СК в момент каждого обращения к подпрограмме *(адрес возврата)*.

Если же подпрограмма использует те же РОН, что и основная программа, сохранению подлежит и содержимое этих регистров. В конце подпрограммы нужно обеспечить восстановление прежнего содержимого как СК, так и этих РОН.

Последовательность возврата в основную программу усложняется, если подпрограмма, в свою очередь, может обращаться к другой подпрограмме и т. д. *(вложенные подпрограммы)*.

Эффективный порядок взаимодействия основной программы и подпрограмм обеспечивает помещение адресов возврата (а если необходимо, и содержимого РОН) в стек, расположенный в свободном месте ОЗУ. Такой стек может быть организован на основе любого РОН. Пример использования стека на базе регистра R5 для сохранения и восстановления содержимого РОН R0...R2 представлен на рисунке 8.2. В этом примере фигурирует **команда пересылки,** относящаяся к классу двухадресных (см. рис. 3.6). Ее мнемоническое обозначение — **MOV** (MOVe), а восьмеричный код $^{0}/_{1}$1SSDD.

Для работы с подпрограммами предусмотрены специальные

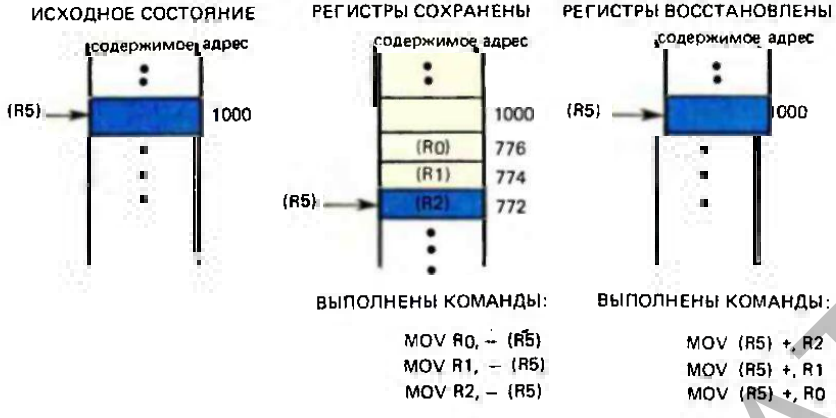


Рис. 8.2. Для работы со стеком, организованным на базе одного из РОН, например R5, можно использовать команду пересылки MOV, автоинкрементный и автодекрементный методы адресации.

команды. **Команда JSR** (Jump to SubRoutine — перейти к подпрограмме) с восьмеричным кодом 004RDD и форматом регистровой команды (рис. 3.7) вызывает занесение содержимого счетчика команд в указанный командой РОН *(регистр связи)*. Предыдущее содержимое этого регистра автоматически помещается в системный стек (на базе R6). Новое содержимое СК (адрес входа в подпрограмму) устанавливается в соответствии с адресной частью команды (DD).

Для организации возврата в основную программу последней командой подпрограммы должна быть **RTS** (ReTurn from Subroutine — вернуться из подпрограммы) с кодом 00020R, вызывающая обратное действие. При этом содержимое регистра связи (R), хранящее адрес возврата, помещается в СК, а «старое» содержимое РОН, используемого в качестве регистра связи, извлекается из системного стека.

Описанный метод гарантирует правильную последовательность действий как в случае вложенных подпрограмм, так и при вызове подпрограммой самой себя *(рекурсия)*.

При вызове подпрограммы в качестве регистра связи можно указать сам счетчик команд R7. В этом случае в системный стек помещается только содержимое СК, т. е. адрес возврата, который считывается при выполнении команды возврата. Соответствующая команда перехода к подпрограмме имеет мнемонический вид:

JSR PC, *subr*,

где PC — стандартное для Ассемблера обозначение *счетчика команд* (Program Counter), *subr* — произвольное символическое обозначение *(метка)* первой команды подпрограммы.

Для передачи значений параметров (аргументов, при которых

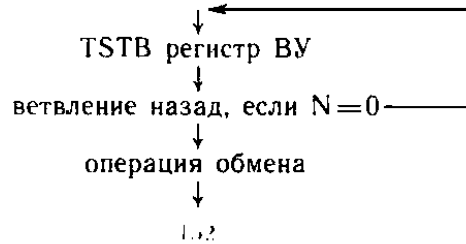
должна выполняться подпрограмма) и возврата результатов в основную программу применяют несколько способов. Самый простой из них состоит в резервировании нужного числа ячеек памяти (или РОН) для совместного использования основной программой и подпрограммой. Иногда удобнее располагать аргументы в ячейках ОЗУ, непосредственно следующих за командой JSR. Так как ее адрес сохраняется в регистре связи, то доступ к параметрам из подпрограммы осуществляют путем автоинкрементной адресации через этот регистр. Если таким же методом извлекать и самый последний параметр, то к моменту исполнения возврата регистр связи будет содержать адрес следующей за аргументами команды программы. Аналогично, вслед за JSR можно помещать адреса аргументов, а доступ к ним иметь посредством косвенной автоинкрементной адресации через регистр связи. В этом случае аргументы могут быть размещены в ОЗУ произвольно. Можно поступить и так: список параметров (адреса или собственно значения) поместить в любом месте ОЗУ, а в подпрограмму передавать только адрес первого элемента списка через любой из РОН.

Очень удобно передавать параметры, помещая их перед входом в подпрограмму в системный стек вместе с информацией о точке возврата.

8.5. Краткий обзор системы команд МП К1801ВМ1

Весь набор команд МП включает 64 команды. Они приведены в Приложении. Многие из них, такие,как INC (INCrement — увеличить на 1), DEC (DECrement — уменьшить на 1), CLR, команды операций с разрядами ССП и другие, не требуют пояснений.

Среди однооперандных команд отметим **команду TST** (от TeST — проверить, код $^0/_{1}057DD$). Ее действие заключается в установке битов N и Z ССП,если операнд (DD) соответственно отрицательный или равный нулю. Так как числа представляются в дополнительном коде, признаком отрицательного числа является установленный старший разряд, и поэтому можно считать, что при исполнении команды TST в разряд N ССП посылается старший бит проверяемого числа, т. е. бит 15 (команда TST) или 07 (команда TSTB). Поскольку в качестве флага готовности ВУ обычно используется бит 07, то команда TSTB весьма удобна для проверки флага. В этом случае за ней должна стоять команда условного перехода назад, если флаг еще не установлен.

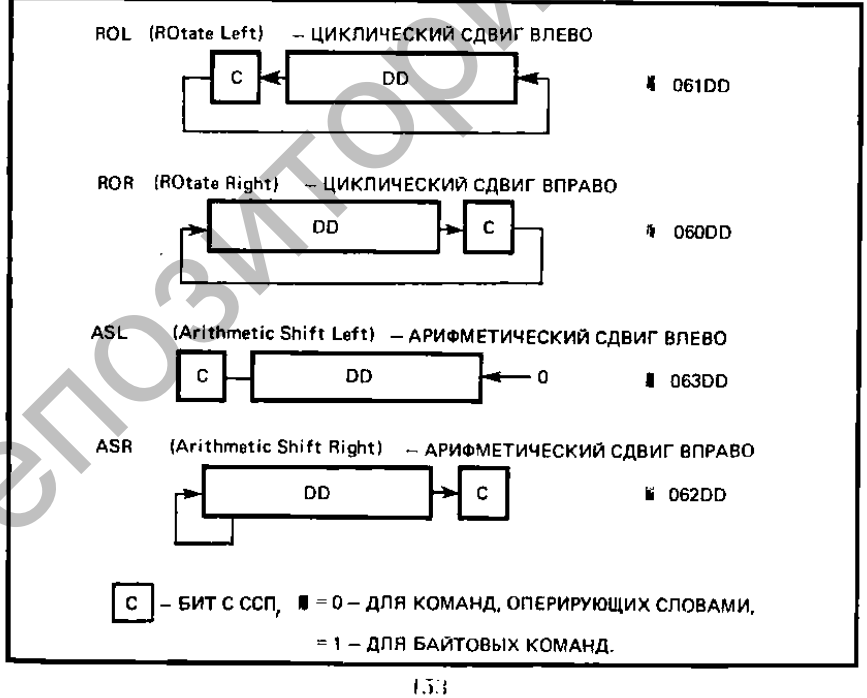


Проверку произвольных разрядов можно произвести двухоперандной **командой BIT** (BIt Test — побитная проверка), которая производит поразрядное логическое умножение двух операндов и заносит информацию о результате в разряды N и Z ССП аналогично команде TST. Сами операнды при этом не изменяются.

Для *сдвигов* операндов существуют четыре специальные команды, действие которых схематично показано на рисунке 8.3. Они широко применяются при умножении и преобразовании двоичных слов данных.

Команды ADC (B), SBC (B) служат для работы с операндами, занимающими более одного слова. Команда ADC (ADd Carry — прибавить перенос) складывает содержимое бита C с указанным операндом, а команда SBC (SuBtract Carry — вычесть перенос) производит его вычитание. В группе двухоперандных команд отметим **команду CMP(B)** (CoMPare — сравнить, код $^{0}/_{1}$2SSDD), сравнивающую значения своих операндов. Фактически она производит вычитание содержимого 2-го операнда (приемника) из 1-го (источника) с выработкой соответствующих результату значений битов ССП, однако после ее выполнения значения обоих операндов сохраняются.

Рис. 8.3. Команды ROL и ROR производят циклический сдвиг операнда на одну позицию через бит C ССП. Команды ASL и ASR — особый вариант сдвиговых операций. Их действие эквивалентно целочисленному умножению (делению) на 2.



Команда SOB (Subtract One and Branch if not equal to 0 — вычесть 1 и перейти, если не равно 0, код 077RNN) исключительно удобна при организации циклов в программе. В начале выполнения команды происходит вычитание единицы из регистра R, затем если результат оказался ненулевой, происходит ветвление назад со смещением NN. Для этого из счетчика команд вычитается удвоенное шестибитное смещение NN.

Группа команд условного перехода *(ветвления,* см. рис. 3.8) включает семнадцать команд, допускающих передачу управления на величину *смещения* (со знаком) в зависимости от значения отдельных битов ССП и их комбинаций. При невыполнении выбранного условия исполняется следующая команда. Поскольку слова имеют четные адреса, а во время исполнения команды ветвления СК уже содержит адрес следующей команды, то для перехода с адреса A на адрес B требуемое смещение можно вычислить по формуле $X = (B - A - 2) : 2$ с записью результата в дополнительном коде. Очевидно, $-128_{10} \leqslant X \leqslant 127_{10} (-200_8 \leqslant X \leqslant \leqslant 177_8)$.

Команды ветвления можно разбить на три подгруппы. В первой (команды BNE, BEQ, BPL, BMI, BVC, BVS, BCC, BCS) анализируются отдельные коды условий ССП. Команды второй подгруппы (BGE, BLT, BGT и BLE) проверяют комбинации кодов условий и позволяют сравнивать операнды с учетом знаков (в дополнительных кодах). Команды третьей подгруппы (BHI, BLOS, BHIS и BLO) применяют для сравнения беззнаковых чисел в натуральном коде. Команды ветвления обладают свойством позиционной независимости.

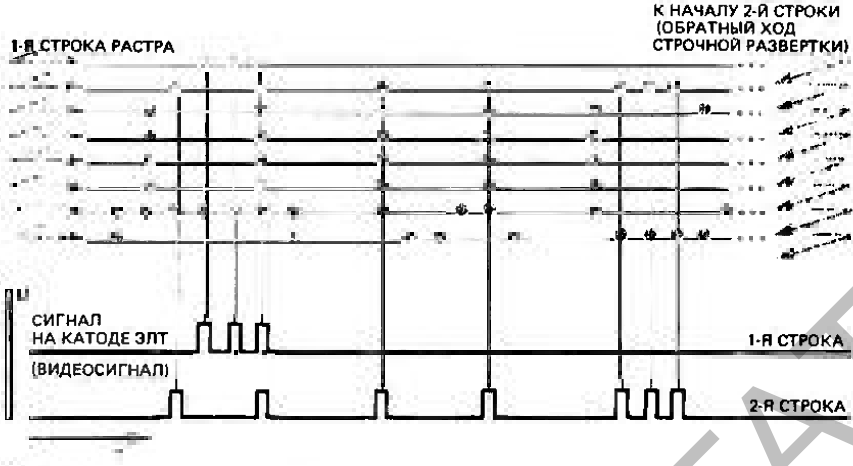
Безадресные команды (рис. 3.10), помимо изменения битов ССП, используются для непосредственного управления процессором. **Команда HALT** (останов, код 000000) прекращает выполнение команд. По **команде WAIT** (ожидание, код 000001) выполнение программы прекращается временно, до поступления прерывания от ВУ. **Команда RESET** (сброс, код 000005) инициирует сигнал INIT на МПИ, используемый для приведения ВУ в исходное состояние. **Команда NOP** (No OPeration, код 000240) не выполняет никакого действия и применяется при отладке программ или организации временных задержек.

8.6. Основные ВУ микро-ЭВМ

Большинство вычислительных систем на основе микро-ЭВМ имеет минимальный набор внешних устройств: алфавитно-цифровой терминал (дисплей), печатающее устройство (принтер) и ВЗУ на гибких магнитных дисках.

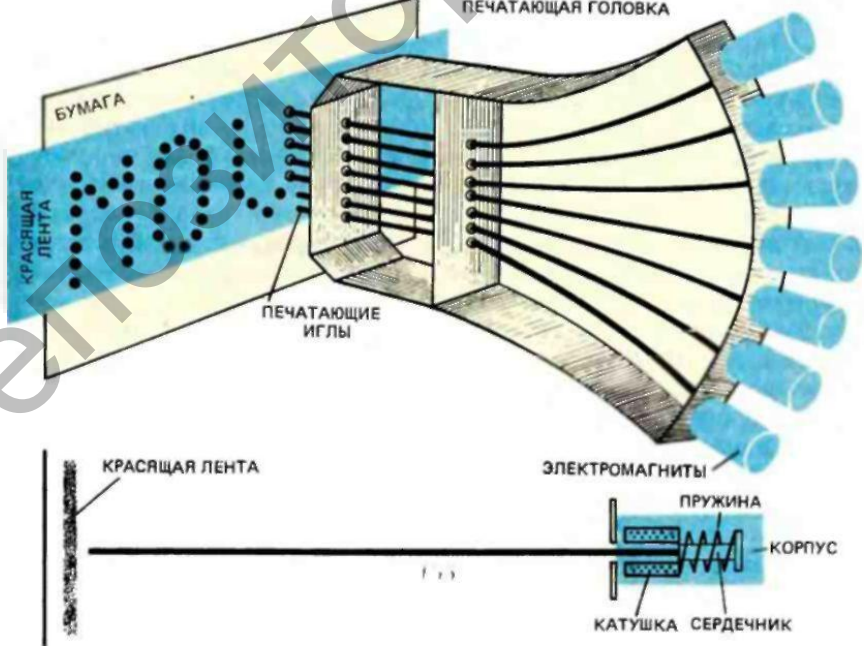
Дисплей, снабженный алфавитно-цифровой клавиатурой, имеет три основных состояния:

1) основное или «комплекс» (on-line) — работа с ЭВМ, при котором клавиатура и экран работают независимо друг от друга, как устройства, соответственно, ввода и вывода;



Вывод одной строки текста на экран алфавитно-цифрового дисплея осуществляется за несколько ходов строчной развертки. Электронным лучом управляет специальная схема знакогенератора, основу которой составляет ПЗУ, содержащее (в двоичном виде) очертания используемых символов.

Печатающая головка принтера содержит вертикальный ряд тонких упругих стержней (игл). В процессе вывода данных из ЭВМ наряду с перемещением головки вдоль печатаемой строки срабатывают выбранные электромагниты, «выталкивающие» иглы к красящей ленте, которая расположена непосредственно перед бумагой. Печатаемые символы обычно формируются в матрице 7×5 точек. При простоте устройства такие принтеры обладают быстродействием до 200 знаков в секунду и в принципе способны к воспроизведению графических изображений.



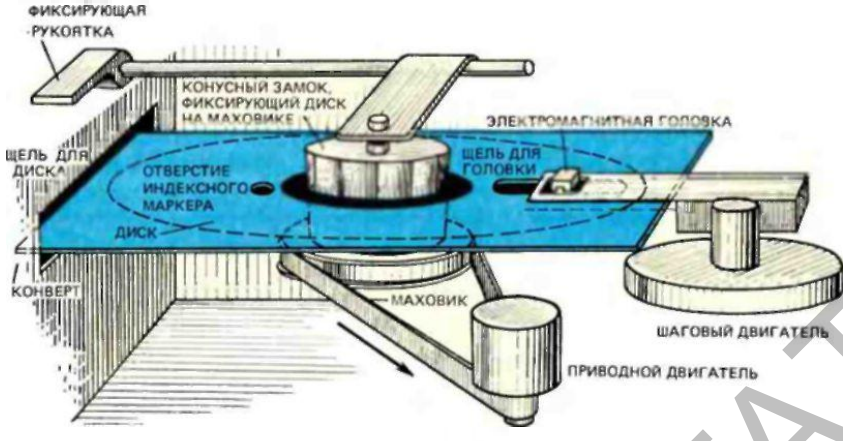


Рис. 8.6. Диск, установленный в НГМД, вращается внутри защитного конверта с частотой 6 оборотов в секунду. Электромагнитная головка может перемещаться вдоль радиуса диска и устанавливаться над одной из дорожек, после чего способна к считыванию/записи информации. Отверстие в диске (индексный маркер) позволяет синхронизировать эти операции с вращением диска посредством простого фотодатчика. Некоторые типы НГМД содержат две головки для работы с верхней и нижней поверхностями диска.

2) «автоном» (off-line) — автономная работа, когда терминал логически отсоединен от ЭВМ и вся информация с клавиатуры попадает непосредственно на экран;

3) «установка» — установка режимов работы терминала.

В режиме «комплекс» при нажатии какой-либо алфавитно-цифровой клавиши формируется соответствующий код, поступающий в регистр данных клавиатуры интерфейса терминала, что сопровождается установкой флага готовности в соответствующем регистре состояния. На экран же информация поступает побайтно из регистра данных экрана, причем коды в интервале 40...177_8 приводят к появлению в очередной позиции экрана соответствующего символа, а коды 0...37_8 являются *управляющими* и не отображаются. Реакция на прием управляющих кодов различна у разных типов терминалов. Для формирования изображений в дисплеях обычно используют растровый метод, аналогичный применяемому в телевизорах (рис. 8.4).

Для получения результатов вычислений, текстов программ и других документов в печатном виде микро-ЭВМ комплектуется *печатающими устройствами* (принтерами). Наиболее распространены точечно-матричные принтеры (рис. 8.5).

Накопитель на гибких магнитных дисках — наиболее распространенное ВЗУ для микро-ЭВМ (рис. 8.6). Гибкие магнитные диски изготовляются из майларовой пленки с ферромагнитным покрытием и имеют диаметр 203 мм, 133 мм и меньше. Информа-

ция записывается на концентрические *дорожки*, каждая из которых делится на *секторы*, содержащие обычно по 128 байт данных. Дорожки располагаются на поверхности диска с плотностью до 60 дорожек/мм, причем каждая из них имеет 8...26 секторов в зависимости от формата записи. Полная емкость одного диска достигает нескольких Мбайт.

8.7. Пультовой режим работы

Большинство микро-ЭВМ не имеет специального пульта управления, при помощи которого оператор может записать или просмотреть содержимое ячеек ОЗУ, запустить или остановить программу, контролировать состояние ЭВМ. Чаще эту роль поручают специальной программе, называемой *эмулятором пульта управления*. Она хранится в специальном системном ПЗУ. «Диалог» с пользователем ведется с помощью алфавитно-цифрового терминала. Обычно микро-ЭВМ входит в режим связи с пультовым эмулятором при включении питания, исполнении команды HALT (останов), появлении активного сигнала на линии аппаратного останова В HALT L и некоторых ошибочных ситуациях.

В этих случаях на экране терминала появляются текущее содержимое СК и символ «@» в новой строке, означающий, что микро-ЭВМ находится в режиме пультового терминала и готова принимать команды оператора. Приведем некоторые из них.

Для просмотра или изменения содержимого ячеек ОЗУ, РОН или ССП следует ввести восьмеричный адрес ячейки, либо наименование РОН (ССП), а затем ввести символ «/» *(«открыть ячейку»)*. Содержимое указанной ячейки, РОН или ССП, выводится в восьмеричном виде вслед за знаком «/». После этого можно ввести новое содержимое открытой ячейки, завершив его одним из следующих символов: <ВК>, <ПС>, «↑», «@» или «_» (подчеркивание). Если новое содержимое ячейки не введено до начала очередной команды, оно останется неизменным.

В любом случае эмулятор закрывает открытую ячейку, а затем, в зависимости от поданной команды, выполняет следующее:

<ПС> — открывает ячейку со следующим адресом;

«↑» — открывает ячейку с предыдущим адресом;

«@» — открывает ячейку, адресом которой является содержимое ранее открытой ячейки или РОН;

«_» — открывает ячейку, адресом которой является содержимое ранее открытой ячейки + ее адрес и + 2;

<ВК> — закрывает открытую ячейку.

Команда «G» вызывает запуск программы. Перед ее подачей необходимо набрать стартовый адрес программы, в противном случае выполнение начнется с адреса 0.

Команда «P» позволяет продолжить выполнение программы,

остановленной по каким-либо причинам. Если при этом на линии B HALT L МПИ постоянно активный уровень, то по каждой пультовой команде «Р» будет выполняться лишь одна команда программы — это используется для пошагового выполнения программ при отладке.

Пример:

```
160000
@
@  1000/177777 240 <ПС>*
001002/177777  123456<ПС>
001004/177777  55<ПС>
001006/000377  <ВК>
```

Включение питания. В ячейках ОЗУ — случайная информация.

Запись в последовательные ячейки ОЗУ чисел 240_8, 123456_8, 55_8 с адреса 1000.

```
@  1000/000240  <ПС>
001002/123456  <ПС>
001004/000055  <ПС>
001006/000377  <ВК>
```

Проверка записанной информации.

```
@  RS /000240  <ВК>
```

Просмотр содержимого ССП.

```
@  R0/120137  <ПС>
R1/000012  <ПС>
R2/000377  <ВК>
@
```

Просмотр содержимого РОН.

Для начальной загрузки ЭВМ используются команды: «177550L» — загрузка с перфоленты; X*n* — загрузка с накопителя на гибких мини-дисках, $n=0...3$ — номер дисковода; D*n* — загрузка с гибких дисков, $n=0,1$ — номер дисковода.

Набор команд пультового режима позволяет вводить в микро-ЭВМ небольшие программы в машинных кодах, отлаживать и выполнять их без применения внешних ЗУ.

Помимо пультового терминала, управление микро-ЭВМ осуществляется выключателем «Питание» и клавишей «Программа/Пульт», изменяющей состояние линии B HALT L магистрали МПИ.

Что такое операционная система?

Составлять программы на понятном для ЭВМ языке ее машинных команд — дело трудоемкое. Еще труднее их модифицировать и отлаживать. Нередко единственная ошибка может потребовать переделки всей программы. Использованное в предыдущих параграфах символическое обозначение машинных команд, бесспорно, упрощает их запоминание и восприятие. С его помощью легче составлять и модифицировать программы, так как подстановка вместо символов их двоичных или восьмеричных эквивалентов может производиться после всех изменений и исправлений. Очевидно, что при строгом соблюдении правил символической записи программы процесс ее перевода в машинные коды носит

---

* Символы, вводимые оператором, выделены шрифтом.

механический характер, а значит, может быть поручен самой ЭВМ. Необходимая для такого перевода *(трансляции)* программа носит название *ассемблер.* Ассемблером называется и язык, на котором составляются символические программы.

Процесс трансляции возможен при наличии хотя бы простейшего ВЗУ для хранения программы-транслятора и терминала для ввода текста программы, воспринимаемой транслятором как исходные данные.

Результат перевода в большинстве случаев целесообразно записать в то же ВЗУ для многократного использования. Роль такого ВЗУ обычно играют НМД (НГМД), реже — НМЛ, а в простейших случаях — *перфоленточные станции,* содержащие фотосчитыватель и перфоратор.

В процессе создания программы неизбежны ошибки, зачастую носящие синтаксический характер. Многие из них может обнаружить хорошо спроектированный транслятор, сообщая об этом посредством терминала. В любом случае избежать повторного ручного ввода исправленной программы позволяет специальная программа — *редактор,* предоставляющая возможность вносить изменения в программы, хранящиеся во ВЗУ. В функции редактора входит также прием исходного текста с терминала и размещение его во ВЗУ.

Процесс разработки прикладных программ упрощают и другие специальные программы: *отладчики,* позволяющие тщательно, шаг за шагом, проконтролировать выполнение прикладной программы; *библиотекари,* накапливающие уже опробованные фрагменты или целые программы универсального назначения,и другие. Все эти программы целесообразно хранить во ВЗУ, извлекая по мере надобности. Процедуры записи и считывания, в общем случае — обмена данными с ВУ, можно поручить специальным программам — *драйверам.* Но остаются заботы типа: где и в какой форме разместить программы во ВЗУ, чтобы избежать путаницы; определить, какие области ВЗУ еще свободны и нет ли в них дефектов; скопировать программу с одного носителя информации на другой; составить список имеющихся программ и т. д. В случае перфоленточных ЗУ эти операции неизбежно требуют достаточно однообразного, рутинного труда. При наличии ВЗУ типа НМД доля этого непроизводительного труда может быть сведена к минимуму. С этой целью используется главная управляющая программа — *монитор,* координирующая работу ВУ, самой ЭВМ и всех специальных программ путем интерпретации команд, подаваемых с клавиатуры терминала.

Монитор, драйверы, трансляторы, редакторы, отладчики — все это компоненты создаваемого высококвалифицированными программистами комплекса программ, называемого *операционной системой* (ОС). Одна из основных функций ОС — обеспечение удобного взаимодействия человека с аппаратурой в процессе разработки программ. «Услуги» операционной системы упрощают

и ускоряют этот процесс, и в результате ОС выступает как своеобразное средство автоматизации программирования.

Монитор или, по крайней мере, основная его часть должны постоянно *(резидентно)* находиться в ОЗУ ЭВМ, вызывая по необходимости остальные компоненты ОС или другие программы. Сам же монитор загружается в ОЗУ с помощью очень короткой *программы начальной загрузки* (bootstrap), хранящейся обычно в ПЗУ ЭВМ.

ОС — не только средство удобного взаимодействия человека с ЭВМ, но и мощный инструмент повышения эффективности использования вычислительной техники, а также среда, поддерживающая создание и выполнение прикладных программ.

8.9. Эволюция операционных систем. Мультипрограммирование

Эффективность использования всех ресурсов ЭВМ высокой производительности с дорогостоящим периферийным оборудованием может быть существенно повышена за счет одновременного коллективного доступа к ним нескольких программ или пользователей.

Коэффициент использования ЭВМ 50-х и даже 60-х годов в среднем не превышал 10 % прежде всего из-за низкой скорости используемых в ту пору ВЗУ на базе перфолент, перфокарт, магнитных лент. Для многих ЭВМ типична занятость множеством различных, непериодически выполняемых программ, в том числе трансляторов, загружаемых с медленного ВЗУ, поэтому процессор значительную часть времени занимается не вычислениями, а вспомогательными операциями ввода-вывода. При этом неизбежны и простои из-за нерасторопности программистов, вынужденных заниматься поиском нужных носителей, установкой их во ВЗУ и т. п. Парадоксально, но достоверно, что с переходом на более быстродействующие процессоры эффективность работы некоторых ЭВМ уменьшалась.

Такая ситуация формировала поиски более быстродействующих ВЗУ. Так появились ЗУ на магнитных барабанах, а затем и дисках. Рутинные же действия операторов ЭВМ постепенно брали на себя все усложняющиеся мониторы — прообразы будущих ОС. Совершенствовались и способы взаимодействия ЭВМ с ВУ: медленно работающие устройства стали снабжаться буферной памятью, а в системе ввода-вывода появился аппарат прерываний. ЭВМ вооружились, например, возможностью «сбросить» в такой буфер целый массив данных, и, не дожидаясь их обработки ВУ, продолжать выполнение программы, которая может быть прервана по мере очередной готовности ВУ к обмену.

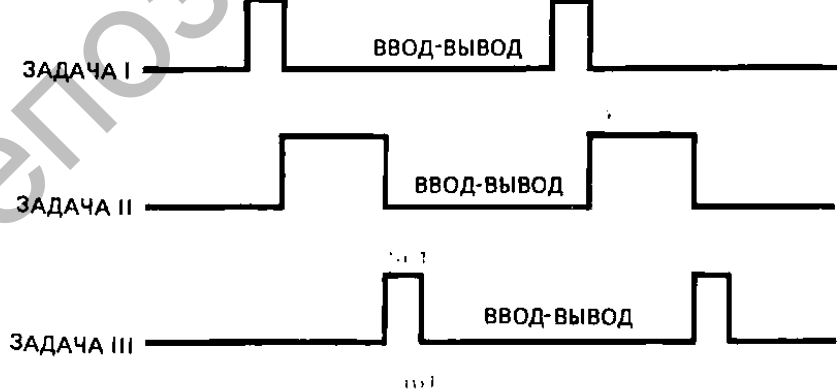
Но далеко не все программы могут продолжать работу, пока не окончился сеанс обмена с ВУ. В результате из совокупности процессора и множества ВУ работающим чаще всего оказывается лишь одно устройство.

Выход был найден в концепции *мультипрограммирования*.

Если в ОЗУ разместить несколько готовых к выполнению программ (задач), то паузой в работе одной программы могут воспользоваться другие. Функцию «переключения» задач естественно поручить монитору, а простои процессора могут быть сведены к минимуму за счет «сеанса одновременной игры» сразу со всеми задачами (рис. 8.7). Еще одна компонента монитора, называемая диспетчером, в состоянии подбирать для ОЗУ такую «смесь» задач из «пакета», установленного на ВЗУ, чтобы таким же образом загрузить почти постоянной работой не только процессор, но и большинство имеющихся ВУ.

В случае параллельного выполнения нескольких программ нельзя полностью «доверить» процессор какой-либо одной из них, поскольку все они должны конкурировать между собой за право доступа к основным ресурсам ЭВМ: ЦП, ВУ и ОЗУ. Чтобы не потерять власть над событиями, монитор должен «перехватывать» в программах команды, претендующие на монопольное использование процессора (например, изменяющие его состояние) или ВУ. Привилегия использования таких команд должна быть уделом только монитора. Проще всего это достигается, если процессор имеет два режима работы: т. н. *привилегированный* (Kernel mode), в котором возможно выполнение всех машинных команд, и *пользователя* (User mode) с ограниченными возможностями для обычных программ. Интересы программистов при этом практически не ущемляются. Они могут использовать в программах многие «запрещенные» команды. Дело в том, что попытка их выполнения процессором приводит к внутреннему прерыванию, вызывающему «на помощь» монитор, который, работая в привилегированном режиме, принимает нужное решение с учетом интересов всех находящихся в ОЗУ задач. Если программа имеет право на требуемое действие, то монитор может произвести его,

Рис. Принцип мультипрограммирования позволяет более эффективно использовать вычислительные ресурсы за счет совмещенного выполнения вычислений и операций ввода-вывода для нескольких задач.



а затем вновь переключить процессор на выполнение программы в режиме пользователя.

Описанный механизм настолько эффективен, что целесообразно обращение пользовательских программ к монитору не только в «принудительном» порядке, но и с обычными запросами, например за услугами по обмену с ВЗУ, к стандартным и всевозможным сервисным программам. Делается это с помощью специальных *директив,* которые существенно расширяют возможности обычных языков программирования, например Ассемблера.

Последовательность выполнения различных задач в мультипрограммных ОС определяется диспетчером. Для рассмотренного нами механизма работы ОС стратегия диспетчера состоит в обеспечении максимальной загрузки всего оборудования задачами, образующими заранее подготовленный пакет. Такой режим работы ОС получил название *пакетного.* Основной недостаток его состоит в том, что ЭВМ «закрыта» для пользователей, которые не имеют возможности вмешаться в выполнение своих задач, даже если оно становится бессмысленным в результате какой-либо ошибки.

Возможность общения программистов с ЭВМ достигается с помощью иных режимов работы ОС, прежде всего — диспетчера. В режиме *разделения временных ресурсов* (РВР), или просто — разделения времени диспетчер поочередно, на равновеликие промежутки *(кванты)* времени предоставляет процессор готовым к исполнению задачам. Кванты имеют величину порядка десятой доли секунды и программист, работающий за терминалом, обычно не замечает, что ЭВМ «отвлекается» на решение других задач. Для ЭВМ эти кванты достаточно велики, и за каждый из них она в состоянии выполнить многие тысячи операций. В результате у каждого программиста создается полная иллюзия работы тет-а-тет с ЭВМ, а стратегию работы диспетчера можно выразить девизом: «всем — поровну».

Такая «уравниловка» неприемлема в целом ряде случаев. Так, в системах автоматизации эксперимента равные права на ресурсы ЭВМ для программ, обеспечивающих накопление и обработку (или вывод) данных, могут привести к потерям измеряемой информации, что недопустимо. Избежать таких ситуаций можно, если присвоить каждой задаче, в зависимости от ее степени важности, определенный *приоритет,* а работу диспетчера подчинить стратегии наискорейшего выполнения задач с наивысшими приоритетами. Такой механизм работы ОС носит название *режима реального времени* (РВ). Недостаток режима РВ — «недемократичность», проявляющаяся в простоях, а иногда — в полном подавлении низкоприоритетных задач при повышенной активности задач с высокими приоритетами.

Современные ОС обычно способны сочетать свойства сразу нескольких режимов работы, благодаря чему недостатки, свойственные каждому из них в отдельности, становятся малосущественными.

Говоря об архитектуре здания, часто имеют в виду пропорции, гармонию — эстетические соотношения между частями единого целого. Специалиста-архитектора интересуют и более конкретные, технические соотношения между подсистемами здания (водо- и электроснабжение, отопление, несущие конструкции и др.). Подобно этому под *архитектурой* сложных систем, в том числе и вычислительных (ВС), понимают способ распределения реализуемых ими функций по определенным уровням организации с точным определением функций, выполняемых на каждом уровне.

С любым уровнем связан определенный язык общения с ВС, т. е. совокупностью аппаратуры и ПО. Заманчиво наделить ЭВМ способностью непосредственно воспринимать предложения языка очень высокого уровня, но проектировать такую машину было бы очень трудно и вряд ли целесообразно. Значительно проще разработать ЭВМ с несложной, но полной системой команд, а на ее основе — ПО, транслирующее в машинные коды команды языка некоторого промежуточного уровня. В свою очередь, на этом языке проще создавать ПО для трансляции предложений языка еще более высокого уровня и т. д. Существуют два механизма трансляции: интерпретация и компиляция.

В процессе работы *интерпретатора* специальная программа распознает очередную команду и осуществляет переход к подпрограмме, выполняющей требуемую функцию. Такая организация служит предпосылкой диалогового режима работы: составляющие интерпретатор программы в состоянии незамедлительно информировать об ошибках в подаче команд, результатах их выполнения, подсказывать правильный порядок действий. Интерпретаторы удобны для обработки команд в ОС, автоматизированных системах управления. Широко распространены интерпретаторы для трансляции и выполнения программ, написанных на языках высокого уровня Бейсик, Фокал.

Трансляторы этого типа имеют и определенные недостатки. Громоздкий интерпретатор (или его часть) должен находиться в ОЗУ вместе с интерпретируемой программой, ограничивая ее размер. Сама же программа выполняется относительно медленно, особенно если она содержит цикл. В этом случае при каждом новом проходе цикла механически повторяется трансляция предложений, образующих тело цикла.

Трансляция методом *компиляции* устраняет эти недостатки. Компилятор транслирует программу целиком, так, что ее выполнение возможно только после трансляции всего исходного текста.

Интерпретаторы обычно содержат в себе все компоненты, необходимые для создания, модификации и отладки программ. Так, существуют версии интерпретатора языка Бейсик, «зашитые» в ПЗУ и способные работать даже на ЭВМ минимальной конфигурации. Компиляторы, которые обычно используют для трансляции с языков типа Ассемблер, Фортран, Паскаль, нужда-

ются в определенном аппаратном и программном обеспечении (см. 8.8).

Традиционно структура архитектурных уровней ВС выглядит следующим образом. Возможности машинного языка развиваются и дополняются на уровне ОС. На базе ОС строятся уровни *языковых процессоров — трансляторов* (интерпретаторов и компиляторов) с различных языков программирования. На этих уровнях ВС обычно сохраняет свойство универсальности в задачах обработки данных. Вышележащие уровни, в числе которых т. н. *пакеты прикладных программ*, создаются на основе подходящего языка программирования и ориентированы на решение уже достаточно узкого круга типовых задач в конкретной (проблемной) области.

Перечисленные уровни образуют иерархическую систему: каждый из них опирается на один из предыдущих, но «обволакивает» его так, что все «винтики» нижележащих уровней обычно оказываются скрытыми от пользователей ВС. Это упрощает не только разработку, но и практическое использование ЭВМ: программисту, особенно начинающему, для работы на Фортране можно ничего не знать о деталях машинного языка и иметь лишь поверхностное представление об уровне ОС.

В ряде же случаев программист обязан знать особенности не только машинного, но и еще более низкого уровня. Дело в том, что для некоторых ЭВМ УУ, входящее в состав процессора, само может быть организовано как некий процессор. Этот процессор, воспринимая код команды «традиционного» машинного уровня, включает для ее выполнения «зашитую» в ПЗУ программу *(микропрограмму)*, состоящую из еще более элементарных, чем команды действий, *микрокоманд* типа занесения в регистр, сдвига, стробирования и т. п. УУ этого типа носят название *микропрограммных*.

Основное преимущество микропрограммных УУ — простота их модификации, что особенно удобно на стадии проектирования. Так, для одного и того же процессора можно разработать несколько систем команд и разместить микропрограммы их выполнения в отдельных ПЗУ. Тогда переход от одной системы команд к другой, т. е. возможность работы с совершенно другой машиной достигается простой заменой ПЗУ в УУ. Такая возможность реализована в некоторых современных ЭВМ для имитации работы других машин. При этом вместо ПЗУ часто используют БИС ЗУПВ, загружаемые в начале работы нужными микропрограммами с внешнего накопителя на гибких магнитных дисках или кассетной ленте. ЭВМ этого типа называются *микропрограммируемыми*.

Аналогичным свойством обладают многие микропроцессоры, пользователям которых предоставляется возможность самим разработать систему команд, наиболее эффективных для решаемой задачи или круга задач. Микропрограммы этих команд и программы, составленные на их основе, после отладки помещаются в отдельные БИС ПЗУ.

Формулировка и уточнение границ архитектурных уровней -

важная часть работы по проектированию ВС. Программисту знание архитектуры ВС необходимо для выбора наиболее эффективных путей и средств решения поставленной задачи.

8.11. Совместимость. Семейства ЭВМ

Быстрый прогресс микроэлектроники приводит к тому, что доля ПО в общем объеме работ по созданию ВС неуклонно возрастает и уже сейчас значительно превосходит долю аппаратного обеспечения. Один из путей снижения трудоемкости создания ПО — достижение *совместимости* различных ЭВМ и ВС, созданных на их основе. Под совместимостью принято понимать возможность выполнения программ, созданных для одной ВС, на других ВС. Совместимость может быть реализована на отдельном архитектурном уровне, например, для всех программ, написанных на языке Фортран. Это достигается стандартизацией требований к разрабатываемым для различных ЭВМ трансляторам. Ведутся работы по стандартизации уровня ОС. Так, распространенная ОС UNIX путем минимальной модификации может быть адаптирована практически к любой ЭВМ. ОС этого типа называют *мобильными*.

Достигнутая на одном из уровней совместимость позволяет заимствовать для новой ВС часть ПО, накопленного в ходе эксплуатации других ВС. При идентичности системы команд двух машин возможен перенос всего ПО одной ЭВМ на другую. В результате можно говорить о совместимости самих машин и о тождественности архитектуры ВС, созданных на их основе.

А правомерно ли в этом случае утверждать о тождественности самих ЭВМ? ЭВМ как сложная техническая система обладает своей архитектурой, включающей микропрограммный, схемотехнический и другие уровни, каждый из которых выполняет свои определенные функции. Для совместимости машин, очевидно, играют роль лишь характеристики самого внешнего уровня, т. е. системы команд. Поэтому под *архитектурой ЭВМ* принято понимать лишь те функции, которые доступны программисту, работающему на уровне машинных команд. Из деталей внутренней организации машины в это понятие входит только то, что каким-либо образом отражается на системе команд: структура памяти, механизмы адресации, наличие и способы использования РОН, особенности системы ввода-вывода и т. п. Таким образом, совместимые ЭВМ обязаны обладать тождественной архитектурой, но могут различаться своим устройством, конструктивным исполнением, потребляемой мощностью, объемом адресуемой памяти, быстродействием и т. д.

Уже по этой причине целесообразен выпуск *семейств ЭВМ*, состоящих из различных модификаций *(моделей)* машин с однотипной архитектурой. Еще большее разнообразие моделей допускает *принцип совместимости «снизу-вверх»*. В этом случае система команд всякой новой (более «старшей») модели обязана

содержать как подмножество базовую систему команд, характерную для самой «младшей» модели, но может расширяться за счет введения новых команд и способов адресации, дополнительных РОН, режимов работы процессора и т. д. В результате на «старшие» модели без каких-либо изменений переносятся все программы, созданные для более «младших» моделей, а дополнительные возможности могут быть использованы для дальнейшего развития ПО.

Отечественной промышленностью выпускается несколько семейств ЭВМ. Семейство ЭВМ *Единой Системы* (ЕС) охватывает диапазон от «больших» высокопроизводительных машин до персональных компьютеров (ЕС 1840, 1841). В *Системе Малых* (СМ) ЭВМ развиваются три архитектурных семейства. В одно из них входит рассмотренный нами МП К1801ВМ1. Основные характеристики этого семейства ЭВМ представлены в таблице 8.1.

Сравнить указанные в таблице модели ЭВМ по производительности довольно сложно, поскольку реальная производительность зависит от целого ряда параметров. Косвенно о производительности можно судить по числу одновременно обслуживаемых пользователей, объему ОЗУ, наличию расширенных наборов команд.

Все, что ранее говорилось о МП К1801, в полной мере справедливо для «младших» моделей ЭВМ в таблице 8.1.

Наиболее серьезное архитектурное отличие «старших» моделей — это увеличение максимально возможного объема ОЗУ до 124 Кслов и до 1920 Кслов с помощью *диспетчера памяти* ММU (от англ. Memory Management Unit — блок управления памятью), который, кроме того, позволяет разделить адресное пространство на секции, ограничивая их влияние друг на друга и обеспечивая тем самым многопользовательский и мультипрограммный режим работы. Что касается программного обеспечения, то наибольшей популярностью на ЭВМ с объемом ОЗУ до 28 Кслов пользуются различные варианты однопользовательской ОС **RT-11** (об этом — следующая глава), а на более производительных моделях применяется, в основном, многопользовательская ОС **RSX-11M** (ОС РВ). В обеих ОС есть возможность выполнять программы, составленные на большинстве широкораспространенных языков программирования: BASIC, PASCAL, FORTRAN-IV, MODULA-2, а для RSX-11M, кроме того, разработаны компиляторы языков FORTRAN-77, FORTRAN-IV PLUS, BASIC-RLUS2, ALGOL, COBOL, C, FORT, RATFOR и другие.

ХАРАКТЕРИСТИКА ПРОГРАММНО-СОВМЕСТИМЫХ МАЛЫХ ЭВМ

Таблица 8.1

ЭВМ	Тип магистрали *	Макс. об'ем ОЗУ, Кслов	Расширенные наборы команд **			Число уровней прерываний	Режим процессора	Стандартное системное ПО	Число пользователей	Тип используемого МП
			EIS	FIS	FPP					
Электроника 100/16	U	28/30	—	—	—	1	KERNEL	РАФОС	1	К581
Электроника 60	Q							ФОБОС (RT-11)		
Электроника МС1201, МС1201.01 ***	Q							ОС ДВК		К1801ВМ1
СМ 3	U									
СМ 1300	U							МОС РВ (RSX-11S)		К1802
Электроника 60М	Q							IDRIS		К581
Электроника 1201.02	Q		+	+	—					К1801ВМ2
Электроника 100/25	U	124				4	KERNEL USER	ОС РВ (RSX—11M)	8—16	
Электроника МС1211	Q							ДОС РВР (RSTS/E)		КН1811
СМ 1300.01	U							ИНМОС (UNIX)		1802/1804
СМ 4	U									
СМ 1600	U			—						
СМ 1420	U,M	1920	+	—	+			RSX—11M/PLUS	до 64	К1804
Электроника МС1212	Q			+				IAS		КН1811
Электроника 79	U,M			+			KERNEL USER SUPERVISOR	UNIX		

Примечания: * Q — магистраль «МПИ» (совмещенные линии адреса/данных;
U — магистраль «Общая шина» (раздельные линии адреса (18 разр.) и данных);
M — магистраль «MASSBUS» (22-разрядная адресная подшина;

** О командах EIS, FIS и FPP см. Приложение

*** Одноплатные микро-ЭВМ «Электроника МС 1201» входят в состав вычислительных комплексов ДВК-1 и ДВК-2.

Глава 9

ВВЕДЕНИЕ В ОС ДВК

К числу распространенных ВС на базе МП К1801 относятся *диалоговые вычислительные комплексы* (ДВК). ДВК содержат дисплей, принтер, НГМД и одноплатную микро-ЭВМ, работающую под управлением операционной системы. Для подобных ВС наиболее хорошо зарекомендовали себя ОС **RT-11** (от англ. Real Time system — система реального времени) и ее разновидности, носящие названия **РАФОС** (РАзделения Функций ОС), **ФОДОС** (Фоново-Оперативная Дисковая ОС), **ОС ДВК** и др. В состав этих ОС, обладающих модульным характером, может входить один из нескольких мониторов, среди которых есть и многозадачные (мультипрограммные). ОС ДВК обычно строится на базе *однозадачного монитора RT11SJ* (Single Job — одно задание).

Эта глава в общих чертах поясняет структуру ОС ДВК, правила использования несложных команд управления системой, работу типичного редактора текстов, этапы разработки программ для ВС.

9.1 Структура ОС ДВК

ОС ДВК состоит из головной управляющей программы (монитора), драйверов внешних устройств, утилитных программ и средств программирования на языках высокого уровня (рис. 9.1).

Монитор — ядро, координирующее работу всех остальных программ, составляющих ОС и осуществляющее связь с оператором (пользователем) через терминал. Богатый набор команд монитора, набираемых пользователем на клавиатуре терминала, по-

зволяет инициировать с их помощью все необходимые действия по управлению системой.

Драйверы внешних устройств — это специализированные программы, предназначенные для управления конкретными периферийными устройствами системы.

Монитор исполняет самостоятельно лишь небольшое количество простейших команд, для выполнения остальных он «призывает на помощь» отдельные системные программы, называемые *утилитными*. В их числе:

1) *редактор текста*, позволяющий ввести, модифицировать и записать на ВЗУ текстовой материал — это может быть программа или любой другой текстовой документ;

2) *утилиты файловой службы*, которые позволяют пересылать данные с одних ВУ на другие, копировать, стирать, обновлять наборы данных и т. д.;

3) *программа-компоновщик*, осуществляющая «сборку» программ из нескольких полученных в результате трансляции двоичных (объектных) модулей.

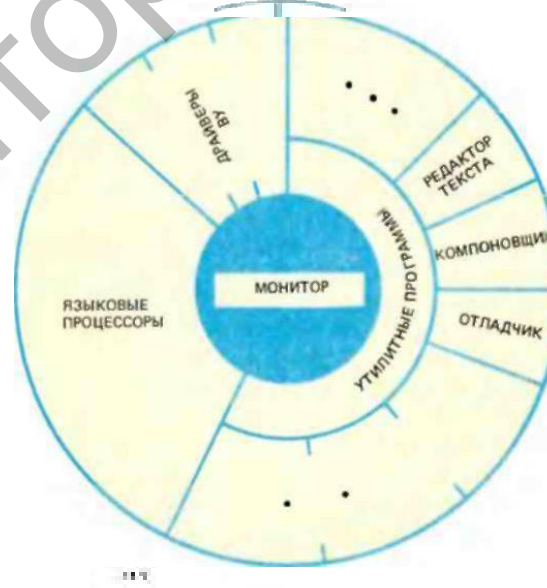
Специальные программы, называемые *языковыми процессорами*, позволяют разрабатывать и выполнять программы, написанные на различных языках программирования.

В системе ОС ДВК могут быть использованы различные пакеты *прикладных программ*: математические, статистические, экономические, системы автоматизации проектирования и т. д.

Все компоненты ОС ДВК хранятся на гибком магнитном диске и вызываются в ОЗУ ЭВМ по мере необходимости. Этим процессом управляет монитор, находящийся в ОЗУ постоянно с момента первоначального запуска системы, т. е. *начальной загрузки*. В ее

Рис. 9.1.

Структура ОС ДВК типична для многих операционных систем — это стройный набор «инструментов» программирования, помогающий создавать сложные и эффективные программы.



ходе нужно установить диск с операционной системой в накопитель и запустить программу начальной загрузки ОС, находящуюся в ПЗУ ЭВМ. По окончании загрузки монитора ОС ДВК на экране видеотерминала появляется сообщение вида:

```
RT11SJ V05.00
.SET TT SCOPE
.
```

Символ . (точка) в левой части экрана в конце сообщения означает, что загрузка окончена, т. е. монитор ОС ДВК находится в ОЗУ ЭВМ и ждет команд пользователя.

9.2. Управление системой. Командный язык

Команды ОС ДВК строятся из английских ключевых слов и аргументов. Для удобства работы большинство ключевых слов можно сокращать до двух-трех первых букв. Каждую команду следует завершать нажатием клавиши ВК (возврат каретки), что инициирует выполнение требуемой функции. До нажатия ВК ошибки при наборе командной строчки можно исправить с помощью клавиш ЗБ (забой — удаление последнего символа в строке) и СУ/U (удаление ошибочно набранной строки целиком). Обозначение типа СУ/U означает, что при нажатой клавише СУ нажимается клавиша U.

Если при вводе команды в ней пропущены какие-либо обязательные элементы или аргументы, то монитор в процессе обработки команды задает пользователю вопросы относительно их значений. Исполнив команду, монитор печатает точку в новой строке, что указывает на готовность принять очередную команду. В качестве примера рассмотрим простейшие команды TIME и DATE, служащие для установки или распечатки, соответственно, текущего времени и даты. Они подаются в следующем виде:

```
.TIME [⟨ЧЧ:ММ⟩]*
.DATE [⟨ЧЧ-МММ-ГГ⟩]
```

Если команды подаются без аргументов, то монитор сообщает текущее время (дату):

```
.TIME 15:00 ⟨ВК⟩**
.DATE 3-OCT-86 ⟨ВК⟩
.DATE ⟨ВК⟩
3-OCT-86
.TIME ⟨ВК⟩
15:00:09
```

---

* Здесь и далее аргументы команд, требующие конкретного задания, и символ возврата каретки заключаются в угловые скобки для отличия от ключевых слов, и дополнительно в квадратные скобки, если аргументы можно опустить.

** Символы, вводимые пользователем, выделены шрифтом.

Файлы и операции с файлами

Наименование файлов и устройств

Обычно данные хранятся на ВЗУ в форме файлов. *Файлом* называется произвольная группа данных, рассматриваемая как единое целое. Это может быть текст программы, программа в двоичном коде, письмо, сообщение, библиотека подпрограмм и т. п. Каждый файл имеет свое уникальное *наименование*, состоящее из системного имени устройства, на котором файл хранится, имени и типа файла, разделенных символом «точка». *Тип файла* может задаваться произвольно, но обычно он несет информацию о содержании файла, например:

SYS — монитор, драйвер, или служебный файл ОС ДВК;
BAS — текст программы на языке BASIC-11;
FOR — текст программы на языке FORTRAN-IV;
MAC — текст программы на языке MACRO-11;
OBJ — двоичный модуль — результат трансляции программы;
SAV — готовая к выполнению программа;
TXT — произвольный текстовой материал и т. п.

Системное имя устройства задается в виде «DD*n*:», где DD — двухбуквенный шифр типа устройства, а *n* — его порядковый номер (указывается лишь при наличии в системе нескольких однотипных устройств), например:

DX0: — накопитель на гибких магнитных дисках номер 0;
MX1: — накопитель на гибких мини-дисках номер 1;
LP: — устройство печати;
TT: — терминал пользователя;
CT0: — накопитель на кассетной магнитной ленте номер 0;
PR: — считыватель с перфоленты;
PP: — ленточный перфоратор;
RK: — накопитель на жестких магнитных дисках емкостью 2,5 Мбайт и т. п.

Помимо этого, в ОС ДВК можно пользоваться *псевдонимами устройств*. Иными словами, с конкретным физическим устройством можно отождествить произвольное имя и затем обращаться к нему по этому имени. Указанная операция осуществляется командой ASSIGN:

.ASSIGN ⟨имя устройства⟩ ⟨псевдоним⟩.

Специальный псевдоним DK: означает «рабочее устройство (диск) пользователя». В наименованиях хранящихся на нем файлов имя устройства может быть опущено (умолчание). В большинстве команд, оперирующих с файлами, допускается употребление вместо имени и/или типа файла символа «*», означающего «любое имя/тип файла», например, запись MX0:*. SYS подразумевает все файлы, хранящиеся на устройстве MX0: и имеющие тип SYS.

Список файлов *(каталог)*, хранимых на ВЗУ, можно получить с помощью команды DIRECTORY. В каталог включается также информация о размере и дате создания каждого файла:

```
.DIR MX0: <BK>
SWAP .SYS  26   19-APR-85 TT       .SYS   2   19-APR-85
DX   .SYS   2   19-APR-85 MX       .SYS   9   19-APR-85
LP   .SYS   2   19-APR-85 DIR      .SAV  17   19-APR-85
DUP  .SAV  21   19-APR-85 PIP      .SAV  16   19-APR-85
LINK .SAV  29   19-APR-85 MACRO    .SAV  45   19-APR-85
LT   .SYS   5   19-APR-85 FORMAT   .SAV  13   19-APR-85
CREF .SAV   6   19-APR-85 DUMP     .SAV   7   19-APR-85
ODT  .OBJ  10   19-APR-85 SYSMAC   .SML  54   19-APR-85
LDA  .SAV   5   19-APR-85 VRF      .SAV   3   19-APR-85
RT11SJ.SYS 70   19-APR-85 SP21     .SAV  21   19-APR-85
K52  .SAV  55   19-APR-85 CAMIN          7   19-APR-85
   22 FILES, 425 BLOCKS
    7 FREE BLOCKS
```

Размер файлов указывается в блоках по 512 байт. По желанию можно получить частичный каталог, включающий в себя один или несколько файлов, имеющих определенное имя и/или тип. Поясним сказанное на примерах:

```
.DIR MX0:* .SYS <BK>
SWAP .SYS  26   19-APR-85 TT       .SYS   2   19-APR-85
DX   .SYS   2   19-APR-85 MX       .SYS   9   19-APR-85
LP   .SYS   2   19-APR-85 LT       .SYS   5   19-APR-85
RT11SJ.SYS 70   19-APR-85
   7 FILES, 116 BLOCKS
   7 FREE BLOCKS
.DIR RT11SJ.* <BK>
RT11SJ .SYS 70 19-APR-85
   1 FILES, 70 BLOCKS
   7 FREE BLOCKS
```

Команда, подаваемая в виде DIRECTORY/PRINTER, позволяет распечатать каталог на печатающем устройстве.

Копирование, переименование, удаление и вывод файлов на печать

Для копирования файлов с одного устройства на другое существует команда COPY, подаваемая в следующем виде:

.COPY/SYS <наименования входных файлов> <наименования выходных файлов>.

Например, команда

```
.COPY MX0: PIP. SAV MX1:
```

создает копию файла PIP .SAV, хранящегося на устройстве MX0:, на устройстве MX1: под тем же именем. Ключ/SYS обязателен при копировании системных файлов, имеющих тип SYS. Например, для создания копии всего диска с операционной системой подается команда

```
.COPY/SYS MX0: *.* MX1:
```

Удаление (стирание) ненужных файлов производится командой .DELETE/NOQ [список стираемых файлов].

При удалении файла запись о нем вычеркивается из каталога и место, отводимое под него на диске, объявляется свободным.

Команда RENAME (переименовать) служит для изменения имени или типа файла:

.RENAME ⟨старое имя файла⟩ ⟨новое имя файла⟩.

В заключение беглого обзора файловых операций приведем команды для вывода содержимого текстовых файлов на экран видеотерминала и на печать:

.TYPE ⟨наименование файла⟩ — вывод файла на экран (TT:);

.PRINT ⟨наименование файла⟩ — вывод файла на устройство печати (LP0:).

9.4. Создание и редактирование текстовых файлов

В текстовых файлах может храниться самая разнообразная информация: программы на каком-либо языке программирования, таблицы, исходные данные для какой-либо программы, списки, письма, напоминание, статьи и т. п. Для создания и редактирования файлов используются редакторы текстов. Наиболее удобны т. н. «экранные» редакторы, ориентированные на работу с определенными типами алфавитно-цифровых видеотерминалов и широко использующие их собственные аппаратные возможности. К ним относится и редактор К52, входящий в состав ОС ДВК.

Редактор К52 создает для пользователя иллюзию, что экран видеотерминала — это «окно» размером 24 строки в редактируемом файле, причем положением «окна» можно управлять с помощью клавиш. Мы не будем рассматривать все функции и возможности К52, поскольку их очень много и они подробно описаны в соответствующей документации, а остановимся лишь на необходимом минимуме.

Запуск редактора текста

Редактирование или создание текстового файла начинается с подачи команды EDIT:

.EDIT/K52/CREATE ⟨наименование файла⟩.

Ключевое слово «/CREATE» информирует К52 о том, что пользователь намерен создать новый файл с указанным в команде наименованием. При отсутствии ключа «/CREATE» подразумевается, что файл уже существует и в нем необходимо сделать изменения или добавления.

При создании файла редактор очищает экран терминала, устанавливает курсор в левый верхний угол экрана, после чего пользователь может начать ввод текстов в файл, завершая каждую строку, как и на обычной пишущей машинке, нажатием клавиши ВК (возврат каретки). В случае, если существующий файл редактируется или дополняется, на экране появляются его первые 24 строки. Все операции по внесению изменений в файл начинаются

с установки курсора в место предполагаемого изменения. Для этого служат клавиши с обозначениями [←], [→], [↑], [↓]*. Нажатие какой-либо из них перемещает курсор на одну позицию в указанном на клавише направлении.

Удаление ошибочно введенного символа

Удаление символов производится клавишами [6] и [ЗБ]. При нажатии первой удаляется символ, указываемый курсором, а нажатие клавиши [ЗБ] удаляет символ слева от курсора (это удобно при вводе текста). При удалении символа правая от курсора часть строки сдвигается влево.

Вставка символов

Для вставки символов никаких специальных действий не требуется. Нажатие клавиши на алфавитно-цифровой клавиатуре вызывает появление символа в позиции курсора, причем текущая строка раздвигается автоматически.

Удаление строки

После установки курсора на строку, подлежащую удалению, требуемая операция производится последовательным нажатием клавиш [0], [↑], [↑]. При этом текст под курсором сдвигается на одну строку вверх.

Вставка строки

Комбинация клавиш [↑] и [0] вызывает появление новой (пустой) строки под текущей позицией курсора. Текст, расположенный ниже курсора, при этом сдвигается вниз.

Перемещение «окна»

В небольших пределах «окно» можно переместить при помощи клавиш управления курсором [↑] и [↓]. Если курсор «упирается» в верхнюю или нижнюю границу экрана, то «окно» автоматически сдвигается. Для более быстрого перемещения «окна» по тексту имеются следующие команды:

[↑], [4] — перемещение «окна» в конец файла;

[↑], [5] — перемещение «окна» в начало файла;

[↑], [↓] — перемещение «окна» вперед-назад на 16 строк.

Направление перемещения в последней команде задается следующими командами:

[4] — установить режим сдвига «вперед»;

[5] — установить режим сдвига «назад».

---

* Все клавиши, применяемые для операций редактирования текста и управления курсором, находятся на служебной клавиатуре видеотерминала справа от основной. Обозначение клавиш соответствует клавиатуре видеотерминала «Электроника 15ИЭ-00-013».

По окончании редакции/ввода текста необходимо выполнить следующую процедуру:

1) последовательно нажать клавиши [↓] и [7], после чего в верхней строке экрана появится приглашение COMMAND;

2) подать команду EXIT, завершаемую нажатием клавиши [↖]. Это вызывает запись отредактированного текста в файл на диске, после чего на нижней строке экрана появляется символ «*»;

3) при нажатой клавише [СУ] нажать клавишу [C]. Работа редактора прекращается и управление передается монитору ОС ДВК.

9.5. **Использование языков программирования**

Существуют сотни языков программирования различных уровней, одни из которых являются проблемно-ориентированными, т. е. предназначенными для решения узкого круга задач в определенной области человеческой деятельности, другие более или менее универсальны. Из языков высокого уровня в ОС ДВК чаще всего используют BASIC и FORTRAN. Универсальный язык низкого уровня (Ассемблер) для рассматриваемых ЭВМ называется **MACRO-11.**

Подготовка программы на языках программирования с трансляторами компилирующего типа, в том числе на MACRO-11 и FORTRAN, включает в себя следующие этапы:

1) составление исходного текста программы на выбранном языке (кодирование программы);

2) ввод текста программы в ЭВМ с помощью редактора текста и запись его в файл на ВЗУ;

3) обработку текста соответствующим языковым процессором и получение в результате двоичного объектного модуля;

4) компоновку готовой к выполнению программы из одного или нескольких объектных модулей;

5) запуск программы.

При использовании компиляторов этапы 2)...4) выполняются посредством отдельных системных программ; если программа пишется на языке BASIC, соответствующие функции осуществляет сам интерпретатор.

Трансляция программы

Трансляция программы, написанной на языке MACRO-11, осуществляется с помощью команды MACRO:

.MACRO ⟨наименование файла с текстом программы⟩.

Пусть программа пользователя хранится на устройстве DK: в файле SUM.MAC. Тогда команда .MACRO SUM приведет к трансляции программы, а ее результат (двоичный *объектный модуль*) будет записан в файл «DK : SUM.OBJ». При желании можно получить, помимо объектного модуля, и листинг, т. е. распечатку программы. В этом случае подаются команды:

.MACRO SUM/LIST:TT: — для выдачи листинга на экран терминала;

.MACRO SUM/LIST — для записи листинга в виде файла с именем SUM.LST на устройство DK:.

Совершенно аналогично записываются команды для трансляции программы на языке FORTRAN-IV:

```
.FORTRAN SUM
.FORTRAN SUM/LIST: TT:
.FORTRAN SUM/LIST
```

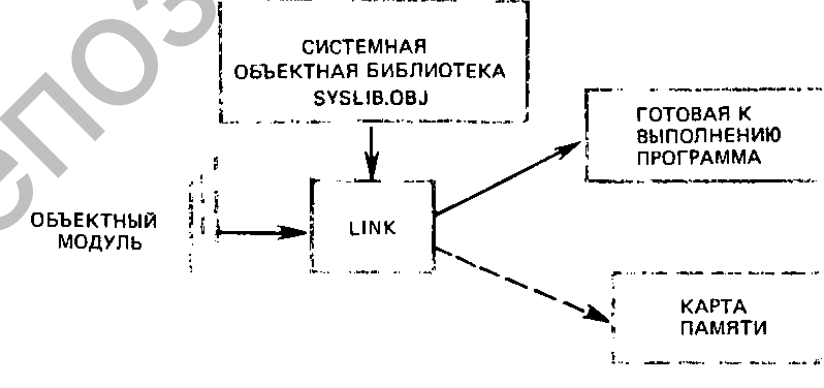
Файл SUM должен иметь тип FOR.

Компоновка и запуск программы

Результат перевода исходного текста компилятором еще не представляет собой готовую к выполнению программу. Во-первых, сложную программу удобно собирать из нескольких модулей исходного текста, создаваемых и транслируемых отдельно. Во-вторых, программа может содержать обращения к хранящимся на ВЗУ библиотекам, содержащим стандартные и системные модули для процедур типа вычисления функций, ввода-вывода и т. д. В-третьих, во многих ОС заранее может быть неизвестно место в ОЗУ, где будет выполняться программа. Таким образом, для получения рабочей программы нужно объединить ее объектные модули, в том числе библиотечные, установив необходимые связи между ними, настроить адреса каждого модуля для выполнения в определенном месте ОЗУ, распределить память под рабочие области и массивы.

Эти и некоторые другие функции осуществляет *программа-компоновщик LINK* (рис. 9.2), команда запуска которого имеет вид:

В задачу утилиты LINK входит создание готовой к выполнению программы из одного или нескольких объектных модулей с автоматическим присоединением, если это необходимо, готовых модулей из системной библиотеки.



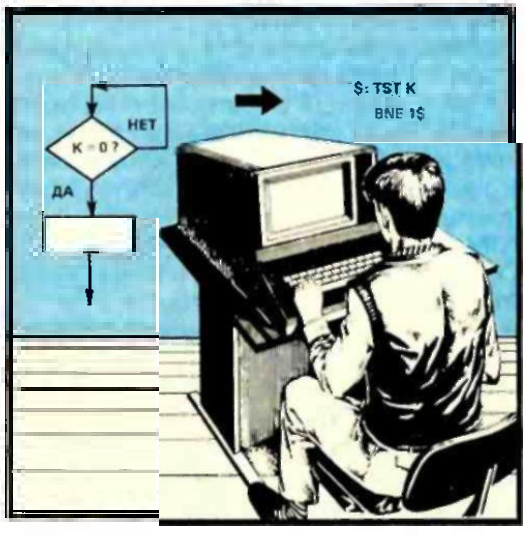
.LINK ⟨список объектных модулей⟩/MAP.

Результатом работы компоновщика является файл, содержащий готовую к запуску и выполнению программу, а если указано ключевое слово /MAP, то и файл, описывающий распределение памяти между отдельными модулями, а также общий объем памяти, необходимый программе («карта памяти»). Выходным файлам присваивается имя первого из указанных в команде объектных модулей, причем файл с программой имеет тип «.SAV», а файл распределения памяти — «.MAP».

Запуск программы на выполнение производится командой .RUN ⟨наименование файла⟩.

Интересная особенность программы LINK — возможность компоновки с перекрытиями, т. е. когда суммарный объем всех модулей написанной программы превышает допустимый верхний предел — около 28 К машинных слов. При этом в ОЗУ помещается не вся программа, а лишь ее основной модуль (корневой сегмент) и необходимые на данный момент подпрограммы, а остальные находятся на ВЗУ и загружаются в ОЗУ по мере необходимости, замещая при этом «отработавшие» модули.

Применение перекрытий в значительной мере снимает ограничение на максимальный размер программы. Недостатки этого механизма — некоторое замедление выполнения программы вследствие затрат времени на загрузку перекрытий с ВЗУ, а также необходимость соблюдения специфической организации программы.



ПРОГРАММИРОВАНИЕ НА АССЕМБЛЕРЕ

Если оператор (команда) языка высокого уровня типа Бейсик или Фортран обычно транслируется в целую группу машинных команд, то сущность *Ассемблера* в символической записи отдельных команд ЭВМ. В результате этот язык полностью отражает архитектуру машины и, следовательно, позволяет создавать наиболее эффективные с точки зрения максимального использования всех возможностей ЭВМ программы.

Важная черта MACRO-11 — наличие механизма макрокоманд. Определяя *макрокоманду*, программист присваивает произвольной последовательности операторов Ассемблера некоторое имя. Появление имени в любом другом месте исходного текста приводит к автоматической подстановке данного определения на этапе трансляции программы. В отличие от подпрограммы, присутствующей в единственном числе как в исходном тексте, так и в его машинном представлении, определение макрокоманды дублируется в машинную программу столько раз, сколько оно вызывается.

Возможность использования макрокоманд, удобных в конкретной задаче или в целом классе задач, приближает MACRO-11 к проблемно-ориентированным языкам высокого уровня.

Строка программы

Исходный текст на Ассемблере записывается в виде последовательности строк, каждая из которых обычно содержит мнемоническое обозначение одной машинной команды. Для того чтобы строка исходного текста была правильно воспринята ассемблером,

необходимо соблюдать правила, совокупность которых называется *синтаксисом языка*.

Строка программы выглядит следующим образом:

МЕТКА: ОПЕРАТОР ОПЕРАНД; КОММЕНТАРИЙ, т. е. содержит 4 поля, которые могут разделяться дополнительными пробелами для того, чтобы сделать программу удобочитаемой, например:

LABEL: ADD R0, R1; ПРИБАВИТЬ R0 к R1

Поле метки, комментарий и операторная часть могут опускаться, причем в произвольных сочетаниях.

Операторы Ассемблера делятся на 3 группы:

1) *описательные операторы* — служат для резервирования внутри программы областей памяти с целью их последующего использования для хранения данных в процессе выполнения программы, присваивания значений переменным и константам и т. п.;

2) *исполняемые операторы* — это собственно машинные команды в мнемонической записи;

3) *управляющие операторы* (директивы) — с их помощью программист осуществляет управление самим ассемблером в процессе трансляции. Мнемоническое обозначение директив всегда начинается с символа точки.

Метка, или символический адрес может состоять из латинских букв, цифр и знака денежной единицы (¤). В процессе трансляции метка приобретает значение, равное адресу, по которому результат перевода обозначенной ею команды будет помещен в память. Это позволяет ссылаться в командах перехода и ветвлений как на численные адреса и смещения, так и на их символические эквиваленты, что гораздо удобнее.

Комментарии в строке программы могут начинаться с произвольного места после оператора или занимать всю строку. Комментарии записываются после символа «точка с запятой».

По умолчанию все числа, имеющиеся в тексте программы, считаются записанными в восьмеричной форме. Для иных представлений используют следующие обозначения:

↑ *Dn* или *n.*— десятичное число *n*,

↑ *Bn* — двоичное число *n*,

↑ *On* — восьмеричное число *n*,

↑ *Cn* — инверсия двоичного представления числа *n*.

Умолчание для восьмеричной формы можно отменить специальной директивой .RADIX.

Запись типа 'X представляет собой константу, равную коду ASCII символа X, а "X_1X_2 обозначает двухбайтовое число, содержащее коды ASCII символов X_1 (младший байт) и X_2(старший байт).

В процессе трансляции программы ассемблер использует специальную переменную, называемую *счетчиком адреса* и обозначаемую символом «.» (точка). Начальное значение счетчика адреса равно 0. В ходе ассемблирования он содержит адрес теку-

щей команды, полученной в результате трансляции. Точка может использоваться для записи адресов в программе. Так, команду перехода на два слова вперед можно записать в виде JMP .+4.

В тексте программы допускается использование *выражений присваивания* типа:

символ (переменная) = выражение.

Выражение может представлять собой число, другую переменную или их произвольную комбинацию, составленную с применением знаков арифметических действий: + (сложение), — (вычитание), × (умножение) и / (деление). Значение выражения подсчитывается слева направо. Важно помнить, что выражения присваивания — это лишь предписания для ассемблера, выполняемые исключительно на этапе трансляции программы.

В ходе трансляции производится синтаксический разбор программы с выдачей сообщений об обнаруженных ошибках. Может быть получен *листинг* — документ, содержащий всю информацию о трансляции, в том числе исходный текст и его машинное представление.

Приведем пример.

```
R50UNP  MACRO V05.00  Monday 01-Dec-86 11:34  Page 1
  1                              .TITLE  R50UNP
  2                     ;+
  3                     ;          ПРЕОБРАЗОВАНИЕ КОДА RADIX-50 В ASCII
  4                     ;  НА ВХОДЕ В П/П :
  5                     ;          R2 -> АДРЕС ASCII СТРОКИ
  6                     ;          SYMBOL, SYMBOL + 2 = 6 СИМВОЛОВ В КОДЕ RAD50
  7                     ;  РЕГИСТРЫ R0, R1 И R3 НЕ СОХРАНЯЮТСЯ
  8                     ;-
  9  000000 010446      R50UNP: :MOV  R4, -(SP)    ; СОХРАНИТЬ R4
 10  000002 012704              MOV  #SYMBOL, R4; УКАЗАТЕЛЬ СИМВОЛОВ RAD50
             000000G
 11  000006 012401      1$ :    MOV  (R4) +, R1    ;  ПОЛУЧИТЬ  ОЧЕРЕДНОЕ  СЛОВО
                                                      RAD50
 12  000010 012703              MOV #50*50, R3     ;  ДЕЛИТЕЛЬ  ДЛЯ  СТАРШЕГО
                                                      СИМВОЛА
             003100
 13  000014 004767              JSR  PC, 10$       ; РАСПАКОВКА И ЗАПИСЬ СИМВОЛА
             000030
 14  000020 012703              MOV #50, R3        ; ДЕЛИТЕЛЬ ДЛЯ СРЕДНЕГО СИМВОЛА
             000050
 15  000024 004767              JSR  PC, 10$       ; РАСПАКОВКА И ЗАПИСЬ
             000020
 16  000030 010100              MOV R1, R0         ; ПЕРЕПИСАТЬ ПОСЛЕДНИЙ СИМВОЛ
 17  000032 004767              JSR  PC, 11$       ; ПРЕОБРАЗОВАТЬ И ЗАПИСАТЬ
             000016
 18  000036 020427              CMP  R4,#SYMBOL   + 4  ; ПОСЛЕДНЕЕ СЛОВО?
             000004G
 19  000042 001361              BNE  1$            ; ПЕРЕХОД, ЕСЛИ ДА
 20  000044 012604              MOV  (SP) +, R4    ; ВОССТАНОВИТЬ R4
 21  000046 000207              RTS  PC            ; ВОЗВРАТ ИЗ П/П
 22                     ;+
 23                     ; ПРЕОБРАЗОВАНИЕ СИМВОЛА В ASCII:
 24                     ;          0 = ПРОБЕЛ          1...32 = A...Z
 25                     ;          33 = $              34     = .
```

```
 26                     ;        35 = НЕИСП.      36. . .47 = 0. . .9
 27                     ; --
 28  000050 005000  10$:    CLR   R0
 29  000052 071003          DIV   R3, R0
 30  000054 005700  11$:    TST   R0
 31  000056 001412          BEQ   23$
 32  000060 020027          CMP   R0#33
             000033
 33  000064 002405          BLT   22$
 34  000066 001402          BEQ   21$
 35  000070 062700          ADD   #22-11, R0
             000011
 36  000074 062700  21$:    ADD   #11-100, R0
             177711
 37  000100 062700  22$:    ADD   #100-40, R0
             000040
 38  000104 062700  23$:    ADD   #40, R0
             000040
 39  000110 110022          MOVB R0, (R2)+
 40  000112 000207          RTS   PC
 41         000001          .END
Errors detected: 0
```

Листинг программы на языке MACRO-11 состоит из следующих полей: 1 — заголовок, 2 — порядковые номера строк программы, 3 — значения счетчика адреса, 4 — коды команд, 5 — исходный текст программы.

10.2. Директивы и описательные операторы ассемблера

Описательные и управляющие операторы (директивы) ассемблера управляют процессом трансляции и не являются обозначениями машинных команд ЭВМ.

Примеры некоторых основных директив и их действия:

.TITLE PUSSYCAT — использовать слово PUSSYCAT в качестве заголовка листинга программы. Эта директива обычно предшествует всей программе;

.WORD ↑B111011, 10., 10,"10 — в данном месте программы резервируется некоторое количество (по числу аргументов) слов памяти, в которые заносятся указанные в директиве константы. В приведенном примере в последовательные ячейки будут занесены следующие восьмеричные числа:

```
000073
000012
000010
030061;
```

.BYTE 35,'A, 150, 40 — подобна директиве .WORD, но резервируются байты, а не слова:

```
040435
020150;
```

.BLKW 10., BLKB 5 — директивы для резервирования указанного числа слов или байтов, заполняемых нулями;

.ASCII /*str*/ — в последовательных байтах размещаются коды символов строки *str*. Вместо символов «/» строка может быть ограничена любыми другими одинаковыми символами (кроме «<» и «=»), которые не встречаются в строке. Непечатные спецсимволы можно записать по их кодам в виде <код>:

```
.ASCII /ДОБРОЕ УТРО/ <15> <12>
.ASCII ?ПОРА ВСТАВАТЬ!? <7> <7>
.ASCII *QUICK BROWN FOX*
```

Директива .ASCIZ действует подобно директиве .ASCII, но дополнительно вставляет нулевой байт в качестве завершающего. Обычно после директив .ASCII, .ASCIZ, .BYTE записывают директиву .EVEN, которая прибавляет единицу к счетчику адреса в случае его возможного нечетного значения.

Последним оператором программы должна быть директива

.END *sym*,

обозначающая конец программы и указывающая стартовый адрес запуска в символическом виде. Например:

.END START — выполнение программы начнется с команды под меткой START.

Для отмены восьмеричного и установки иного представления чисел, действующего по умолчанию, используется директива .RADIX *n*, где *n*=2, 10 или 8.

10.3. Как писать программы на Ассемблере?

Записи программы на Ассемблере должна предшествовать тщательная и детальная разработка алгоритма. Чтобы избежать путаницы при составлении даже простейших программ, удобно прежде всего составить блок-схему, а лучше — несколько блок-схем, различающихся степенью детализации алгоритма. При этом на самой первой схеме обычно изображаются общие этапы работы программы, а на последней — детально описывается каждый шаг, вплоть до отдельных машинных команд. Опыт говорит о том, что при наличии детальной блок-схемы процесс написания программы на Ассемблере существенно облегчается, ускоряется и требует лишь внимания и аккуратности.

Кроме того, исключительно важно знание структуры процессора (хотя бы в общих чертах) и машинного языка (архитектуры) используемой ЭВМ. Действительно, если программист не совсем точно представляет себе, каким образом ассемблер истолкует строку программы, переведет ее в машинный код, и как соответствующая ему команда выполняется, то вряд ли можно надеяться на правильную работу программы.

Вот несколько примеров, поясняющих сказанное. Допустим, желательно использовать индексный метод адресации с автоматическим увеличением содержимого регистра. Мнемоническое обо-

значение команды, использующей подобную операцию, могло бы выглядеть следующим образом:

```
CLRB 1000(R0)+
```

Вместе с тем указанного метода адресации не существует. К счастью, ассемблер тщательно анализирует допустимость применяемых методов адресации и информирует о совершенных ошибках. С другой стороны, оператор типа JMP R0 транслируется в команду с кодом 000100 без какого-либо сообщения, хотя нулевой метод адресации в сочетании с командой JMP недопустим и при ее выполнении вызывает прерывание по вектору 4. При необходимости передать управление по адресу, хранящемуся в R0, следует использовать метод 1: JMP (R0). Еще пример. Допустим, к содержимому РОН R0 нужно прибавить 4, а результат отправить в R1. Здесь легко допустить грубую ошибку, которая не будет зарегистрирована ассемблером:

```
MOV R0+4, R1
```

В результате трансляции получим инструкцию, эквивалентную

```
MOV R5, R1,
```

а требуемая операция правильно реализуется двумя командами:

```
MOV R0, R1
ADD # 4, R1
```

Следует обращать особое внимание на используемые режимы адресации для обращения к адресам и константам, поскольку, например, две следующие команды выполняют совершенно разные операции:

```
1) MOV # 1000,R0
2) MOV 1000,R0
```

В первом случае в регистр R0 будет записана константа 1000_8 (непосредственная адресация), а во-втором — содержимое ячейки с адресом 1000_8 (относительная адресация).

Следует четко разграничивать операции, выполняемые ассемблером в ходе трансляции, и выполняемые процессором при работе транслированной программы. Так, существует большая разница между командами

```
LOC:.WORD 0
```

и

```
CLR LOC
```

Первая из них предписывает ассемблеру зарезервировать ячейку памяти и заполнить ее константой 0, а вторая обозначает команду процессора, выполняемую во время работы программы.

Наконец, приведем текст программы, которую читателям

предлагается самостоятельно перевести на машинный язык и проанализировать:

```
        .TITLE   EXAMPLE
        .= 1000
START:  MOV      #A3,R0          ; УСТАНОВИТЬ АДРЕС СТРОКИ
        MOV      #6,R1           ; СЧЕТЧИК ЦИКЛОВ
LOOP:   MOVB     (R0)+,R2        ; ИЗВЛЕЧЬ ОЧЕРЕДНОЙ СИМВОЛ
        BIC      #177770,R2      ; ОЧИСТКА НЕНУЖНЫХ РАЗРЯДОВ
        ASL      R3              ; СДВИГ
        ASL      R3              ; СДВИГ
        ASL      R3              ; СДВИГ
        ADD      R2,R3           ; ДОБАВЛЕНИЕ ОЧЕРЕДНОЙ ТРИАДЫ
        SOB      R1,LOOP         ; ПОВТОР 6 РАЗ
        MOV      R3, (PC)        ; ПЕРЕСЫЛКА ДЛЯ ВЫПОЛНЕНИЯ
        .WORD    0               ; МЕСТО ДЛЯ КОМАНДЫ
        HALT
A3 :    .ASCII   "010704"
        .END     START
```

Что содержится в регистрах R0...R4 по окончании работы программы?

10.4. **Подпрограммы**

Необходимая информация, касающаяся вызова и возврата из подпрограммы, уже изложена в главе 8. Напомним способы передачи аргументов в подпрограммы и способы пересылки результатов. Выбор метода передачи аргументов в подпрограммы зависит от количества аргументов и удобства применения того или иного метода в конкретном случае. При небольшом числе аргументов их можно передать в регистрах R0...R5 и через них же получить результат. При значительном количестве аргументов их размещают в последовательных ячейках ОЗУ и загружают в какой-либо из свободных РОН начальный адрес полученного массива.

Наиболее популярный метод — передача аргументов через стек. Перед входом в подпрограмму аргументы загружаются в стек, организованный на регистре R6, а по окончании ее работы стек очищается. Необходимо обратить внимание на то, что адрес возврата помещается в тот же стек самым «нижним» элементом, и следует позаботиться о его сохранности.

В качестве примера рассмотрим программу умножения двух целых двоичных 16-разрядных чисел без знака.

```
        .TITLE   MULT
START:  MOV      #58.,R0         ; УСТАНОВИТЬ СОМНОЖИТЕЛИ
        MOV      #30.,R1         ; В R0 И R1
        JSR      PC, MULT        ; ВЫЗОВ П/П УМНОЖЕНИЯ
        HALT
```

```
;+
;          ПОДПРОГРАММА УМНОЖЕНИЯ 16-РАЗРЯДНЫХ
;                       ЦЕЛЫХ ЧИСЕЛ
;-
MULT:   MOV     R0, R2      ; МНОЖИМОЕ -> R2
        CLR     R0
        MOV     #16.,R3     ; УСТАНОВИТЬ СЧЕТЧИК ЦИКЛА
        ROR     R1          ; СДВИГ СОМНОЖИТЕЛЯ (МЛ. БИТ R1 ->
                            ; С-БИТ ССП)
LOOP:   BCC     L           ; ПЕРЕХОД, ЕСЛИ  C=0
        ADD     R2, R0      ; C=1 --> ДОБАВИТЬ МНОЖИМОЕ К R0
        CLC                 ; ОЧИСТИТЬ С-БИТ ССП
L:      ROR     R0          ; ЦИКЛИЧЕСКИЙ СДВИГ
        ROR     R1          ; ЧЕРЕЗ С-БИТ ССП
        SOB     R3, LOOP    ; ПОВТОР 16 РАЗ (SOB НЕ ИЗМЕНЯЕТ С-БИТА)
        RTS     PC          ; ГОТОВО -- ВОЗВРАТ ИЗ П/П
        .END    START
```

Результат программы представляет собой 32-разрядное число, т. е. двойное машинное слово. Умножение выполняется с помощью операций сдвига и сложения. Подпрограмма умножения, составляющая основу программы, возвращает результат в регистрах R1 (младшая часть) и R0 (старшая часть). В них же при входе в подпрограмму находятся два 16-разрядных сомножителя. Последовательность действий при умножении выглядит следующим образом (случай 6-разрядных чисел):

$58_{10} = 72_8 =$ 111010_2 (множимое)
$30_{10} = 36_8 =$ $\times$ 011110_2 (множитель)

```
        000000   слагаемые 12-разрядного
      + 111010   произведения формируются
      + 111010   путем сдвига множимого
        111010
       111010
------------------
110111001100 = 1740(10).
```

10.5. **Программирование ввода-вывода**

В простейших программах, рассмотренных до сих пор, результаты размещались либо в ОЗУ, либо в РОН, а исходные данные записывались с использованием непосредственного метода адресации. На практике так поступают довольно редко, потому что обычно программы составляются для неоднократного использования, исходные данные могут меняться, а результаты необходимо напечатать или же записать во ВЗУ.

Остановимся на программировании операций ввода-вывода информации для двух простейших устройств — терминала и перфоленточной станции. Простоту операций с регистрами внешних устройств как с обыкновенными ячейками ОЗУ в ЭВМ типа ДВК наглядно демонстрирует программа вывода на экран символов, набираемых с клавиатуры. (Напомним, что для ЭВМ клавиатура и экран — это независимые устройства.)

```
1                                       .TITLE  ECHO
2                               ;+
3                               ;       ПРОГРАММА ПЕРЕСЫЛКИ ДАННЫХ С КЛАВИАТУРЫ
4                               ;       НА ЭКРАН ТЕРМИНАЛА (ЭХО-ПЕЧАТЬ)
5                               ;-
6                    177560             TKS = 177560    ;   РЕГИСТР СОСТОЯНИЯ КЛАВИАТУРЫ
7                    177562             TKB = 177562    ;   РЕГИСТР ДАННЫХ КЛАВИАТУРЫ
8                    177564             TPS =  177564   ;   РЕГИСТР СОСТОЯНИЯ ЭКРАНА
9                    177566             TPB =  177566   ;   РЕГИСТР ДАННЫХ ЭКРАНА
10
11   000000  105737  ECHO:   TSTB   @#TKS;   КЛАВИША НАЖАТА?
             177560
12   000004  100375          BPL    ECHO    ;   НЕТ – ПЕРЕХОД ОБРАТНО
13   000006  105737  LOOP:   TSTB   @#TPS;   ЭКРАН ГОТОВ?
             177564
14   000012  100375          BPL    LOOP    ;   НЕТ – ПЕРЕХОД ОБРАТНО
15   000014  113737          MOVB   @#TKB,@#TPB    ; ПЕРЕСЫЛКА БАЙТА
             177562
             177566
16   000022  000766          BR     ECHO    ;   ... И ВСЕ СНАЧАЛА
17           000000'         .END   ECHO
```

Обратите внимание на эффектный способ анализа состояния флага готовности в регистрах TKS и TPS. Если рассматривать младший байт этих регистров как некоторое число, то бит 07 представляет его знак ($0 = +$, $1 = -$) и записывается в бит N ССП при выполнении команды TSTB. Затем его значение анализируется командой BPL, которая осуществит ветвление, если бит N ССП равен 0.

В качестве следующего примера рассмотрим вывод строки текста на экран терминала.

```
        .TITLE  WRLINE
;+
;       ПРОГРАММА ПЕЧАТИ СТРОКИ ТЕКСТА
;           НА ЭКРАНЕ ТЕРМИНАЛА
;-
        TPS = 177564        ; ОПРЕДЕЛЕНИЕ АДРЕСОВ
        TPB = 177566        ; РЕГИСТРОВ ТЕРМИНАЛА
;
START:  MOV     #BUFFER, R0         ; УСТАНОВИТЬ АДРЕС СТРОКИ
LOOP:   TSTB    @#TPS               ; ЭКРАН ГОТОВ?
        BPL     LOOP                ; ЕСЛИ НЕТ -- ПЕРЕХОД ОБРАТНО
        MOVB    (R0)+,@#TPB         ; ВЫВОД ОЧЕРЕДНОГО СИМВОЛА
        TSTB    (R0)                ; СЛЕДУЮЩИЙ СИМВОЛ НУЛЕВОЙ?
        BNE     LOOP                ; ЕСЛИ НЕТ -- ПЕРЕХОД ОБРАТНО
        HALT                        ; ДА -- ОСТАНОВ
BUFFER: .ASCIZ  "*****ПРИВЕТ*****"
        .END    START
;
```

Для вывода строки текста на экран программа циклически опрашивает готовность терминала к приему данных и при установленном флаге посылает очередной символ в регистр данных экрана. Регистр R0 используется в качестве указателя на очередной байт в выводимой строке. Читателям предлагается самостоятельно проанализировать работу этой программы.

Попытаемся разобраться с выводом цифровых данных. Составим программу для распечатки содержимого регистра R0 в восьмеричной форме. Взглянув на таблицу алфавитно-цифровых кодов, можно заметить, что алфавитно-цифровой код цифры в интервале 0...7 легко получить путем прибавления к ней числа 60_8. Например, если в R1 содержится число 7_8, то, прибавив к R1 60_8 и выдав его младший байт на экран, мы увидим цифру 7. Поэтому программа должна «разбить» содержимое R0 на тройки бит, каждая из которых соответствует одной восьмеричной цифре, добавить к каждой тройке 60_8 и распечатать их в нужном порядке. Программа выглядит так:

```
        .TITLE  PRINTR0
;+
;       ПРОГРАММА РАСПЕЧАТКИ СОДЕРЖИМОГО
;       РЕГИСТРА R0 В ВОСЬМЕРИЧНОМ ВИДЕ
;               НА ЭКРАНЕ ТЕРМИНАЛА
;-
        TPS = 177564            ; ОПРЕДЕЛЕНИЕ АДРЕСОВ
        TPB = 177566            ; РЕГИСТРОВ ТЕРМИНАЛА
;
START:  CLR     R1              ; R1 БУДЕТ СОДЕРЖАТЬ ВЫВОДИМЫЙ СИМВОЛ
        JSR     PC, EP2         ; ПЕЧАТЬ ПЕРВОЙ ЦИФРЫ (0 ИЛИ 1)
        MOV     #5, R2          ; СЧЕТЧИК ЦИКЛА
LOOP:   JSR     PC, EP1         ; ПЕЧАТЬ ОЧЕРЕДНОЙ ЦИФРЫ (0...7)
        SOB     R2, LOOP        ; ЦИКЛ ОКОНЧЕН?
        HALT                    ; ДА -- ОСТАНОВ
;+
;       ПОДПРОГРАММА ФОРМИРОВАНИЯ КОДА
;               И ПЕЧАТИ ЦИФРЫ
;-
EP1:    CLR     R1              ; ФОРМИРОВАНИЕ В R1
        ROL     R0              ; ОЧЕРЕДНОЙ ТРОЙКИ БИТ,
        ROL     R1              ; СООТВЕТСТВУЮЩЕЙ ОДНОЙ
        ROL     R0              ; ВОСЬМЕРИЧНОЙ ЦИФРЕ
        ROL     R1              ; (СДВИГ ЧЕРЕЗ С-БИТ ССП)
EP2:    ROL     R0              ;
        ROL     R1              ;
        ADD     #'0,R1          ; ДОБАВИТЬ К R1 КОД СИМВОЛА "0"
;+
;       ВЫВОД НА ЭКРАН СИМВОЛА ИЗ R1
;-
LOOP1:  TSTB    @#TPS           ; ЭКРАН ГОТОВ?
        BPL     LOOP1           ; ЕСЛИ НЕТ -- НАЗАД
        MOVB    R1@#TPB         ; ВЫВОД СИМВОЛА
        RTS     PC              ; ВОЗВРАТ ИЗ П/П
        .END    START
```

Попробуйте самостоятельно написать программы, осуществляющие функции, обратные последним двум программам, т. е. программу ввода текста в ОЗУ с клавиатуры терминала и программу ввода восьмеричного числа в R0.

10.6. Прерывания программы

Большинство относительно медленно работающих устройств — печатающих, ввода-вывода с перфоленты и т. п.— программируется способами, несущественно отличающимися от только что рассмотренных. Следует лишь помнить, что адреса регистров данных и состояния индивидуальны для каждого конкретного устройства. Высокоскоростные устройства внешней памяти на магнитных лентах и дисках программируются несколько иначе, потому что пересылка данных между ВЗУ и ОЗУ осуществляется с помощью прямого доступа в память (ПДП) блоками слов (чаще всего по 256). Процессор лишь указывает контроллеру ВЗУ номер дорожки и сектор магнитного диска, направление пересылки данных и начальный адрес отведенного для пересылаемой информации буфера в ОЗУ через специально выделенные для ВЗУ регистры команд и управления/состояния.

Основная программа обязана предварительно занести в используемые векторы прерываний нужные стартовые адреса подпрограмм обработки прерываний и значения ССП, и лишь затем позволить внешним устройствам осуществлять прерывания. Для разрешения/запрещения прерываний принято использовать разряд 06 регистров состояния внешних устройств.

Приведем пример программы дублирования перфолент, работающих с использованием прерываний.

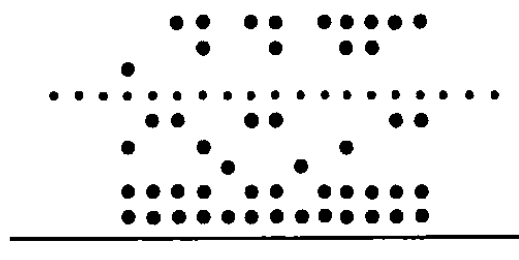
```
        .TITLE  INTREX
;+
;       ДЕМОНСТРАЦИОННАЯ ПРОГРАММА, ПОЯСНЯЮЩАЯ
;       ПРОГРАММИРОВАНИЕ ВУ С ИСПОЛЬЗОВАНИЕМ
;       ПРЕРЫВАНИЙ ПРОГРАММЫ
;-
        PRS = 177550        ; РЕГИСТР СОСТОЯНИЯ СЧИТЫВАТЕЛЯ
        PRD = 177552        ; РЕГИСТР ДАННЫХ СЧИТЫВАТЕЛЯ
        PPS = 177554        ; РЕГИСТР СОСТОЯНИЯ ПЕРФОРАТОРА
        PPD= 177556         ; РЕГИСТР ДАННЫХ ПЕРФОРАТОРА
        PRV = 70      ; АДРЕС ВЕКТОРА ПРЕРЫВАНИЙ СЧИТЫВАТЕЛЯ
        PPV = 74      ; АДРЕС ВЕКТОРА ПРЕРЫВАНИЙ ПЕРФОРАТОРА
;
START:  MOV     #PPISR,@#PPV        ; ЗАНЕСЕНИЕ ИНФОРМАЦИИ
        MOV     #PRISR,@#PRV        ; В ВЕКТОРЫ ПРЕРЫВАНИЙ
        CLR     @#PPV + 2           ;
        CLR     @#PRV + 2           ;
        MOV     #101,@#PRS          ; РАЗРЕШЕНИЕ ПРЕРЫВАНИЙ
                                    ; И "СТАРТ" СЧИТЫВАТЕЛЯ
```

```
                    .                    ;
                    .                    ; ОСНОВНАЯ ПРОГРАММА
                    .                    ;
                                         ;
; +
;       ПРОГРАММА ОБСЛУЖИВАНИЯ ПРЕРЫВАНИЙ СЧИТЫВАТЕЛЯ
; -
PRISR:  MOVB  @#PRB,@#PPB,       ; ПЕРЕСЫЛКА БАЙТА
        MOV     #100,@#PPS       ; РАЗРЕШЕНИЕ ПРЕРЫВАНИЙ
                                 ; ПЕРФОРАТОРА
        RTI                      ; ВОЗВРАТ ИЗ ПРЕРЫВАНИЯ
; +
;       ПРОГРАММА ОБСЛУЖИВАНИЯ ПРЕРЫВАНИЙ ПЕРФОРАТОРА
; -
PPISR:  CLR   @#PPS        ;ЗАПРЕТ ПОВТОРНЫХ ПРЕРЫВАНИЙ
        MOV     #101,@#PPS       ; РАЗРЕШЕНИЕ ПРЕРЫВАНИЙ
                                 ; СЧИТЫВАТЕЛЯ, "СТАРТ"
        RTI                      ; ВОЗВРАТ ИЗ ПРЕРЫВАНИЯ
        .END   START
```

Перфолента представляет собой бумажную полосу, данные на которую записываются путем пробивки групп отверстий, расположенных определенным образом (рис. 10.1). Перфорирование или считывание осуществляют последовательно по 1 строке за цикл, т. е. по 1 байту данных. Порядок работы с регистрами перфоленточной станции почти такой же, как и в случае терминала, и отличается лишь тем, что для считывания очередного байта данных с перфоленты необходимо подать команду «пуск фотосчитывателя» путем установки в «1» младшего бита регистра состояния считывателя.

Использование прерывания — основной метод работы ВУ в сложных программных системах. В этом случае подпрограммы обслуживания ВУ можно оформить в виде самостоятельных модулей, выполняющихся асинхронно (т. е. независимо во времени), по отношению к основному модулю. Такие модули управления работой ВУ называются *драйверами*.

Рис. 10.1. Участок перфоленты.



10.7. **Взаимодействие программ с ОС**

Обычно программы выполняются «в среде» операционной системы (ОС). Иными словами, в момент выполнения программы помимо нее в ОЗУ ЭВМ, как правило, присутствует монитор ОС, одна из функций которого — управление вводом-выводом. Поэтому при работе с ОС в приведенных ранее примерах все манипуляции с регистрами ВУ следует заменить на обращения к ОС. Такого рода обращения осуществляются с помощью специальной *команды программного прерывания* EMT в строго определенном порядке. Вместе с тем, запоминать порядок обращения к ОС для получения того или иного обслуживания нет необходимости благодаря мощному аппарату системных макрокоманд Ассемблера. Системная макрокоманда — это совокупность машинных команд, предварительно определенная в специальной (системной) библиотеке и вызываемая автоматически по имени. Перед использованием макрокоманд в программе должна быть директива

.MCALL *имя 1, имя 2, имя 3, ...,*

которая объявляет, что указанные в ней имена являются именами макрокоманд, описанных в системной макробиблиотеке.

Для работы с терминалом в ОС ДВК существуют макрокоманды .TTYOUT и .TTYIN, предназначенные для вывода на экран и ввода с клавиатуры одного байта данных.

В качестве параметра при вызове этих макрокоманд указывают символический адрес или обозначение РОН, куда следует направить или откуда взять пересылаемый байт. Кроме того, при работе с ОС команду HALT следует заменить макрокомандой .EXIT, которая передает управление монитору ОС по окончании выполнения программы. В противном случае произойдет останов процессора и потребуется перезагрузка ОС для продолжения работы.

Пример программы для работы в среде ОС:

```
        .TITLE  OUTSTR
;+
;       ПРОГРАММА ВЫВОДА СТРОКИ ТЕКСТА НА ЭКРАН ТЕРМИНАЛА
;-
        .MCALL  .TTYOUT,.EXIT       ; ВЫЗОВ МАКРОКОМАНД
START:  MOV     #MSG,R0             ; НАЧАЛЬНЫЙ АДРЕС СООБЩЕНИЯ
        MOV     #ENDMSG-MSG,R1      ; ДЛИНА СООБЩЕНИЯ
LOOP:   .TTYOUT (R0)+               ; ВЫВОД СИМВОЛОВ
        SOB     R1,LOOP             ; В ЦИКЛЕ
        .EXIT
MSG:    .ASCII  <15><12>"ВВОД-ВЫВОД В ОС ДВК"
ENDMSG:
        .END    START
```

Конкретный набор и имена макрокоманд строго индивидуальны для каждой ОС. Так, в многопользовательской ОС RSX-11M для всех операций ввода-вывода используется универсальная макрокоманда QIOW$.

Перемещение и компоновка. Позиционно-независимое кодирование

Как уже отмечалось, этапу выполнения программы предшествует ее компоновка. *Компоновщик* осуществляет две основные функции:

1) модифицирует адреса и константы для «настройки» двоичной программы на конкретные физические адреса ОЗУ;

2) согласует взаимные обращения между различными модулями.

Необходимость в модификации адресов возникает потому, что исходная программа на Ассемблере всегда транслируется с учетом предполагаемого размещения с адреса 0. В процессе компоновки каждый объектный модуль получает некоторый участок адресного пространства, и все адреса, а также некоторые константы модифицируются с учетом базового адреса этого участка. В ряде случаев один модуль может обращаться к ячейкам, находящимся в другом. При этом метка соответствующей ячейки должна определяться в одном и другом модуле как *«глобальная»* с помощью директивы:

.GLOBL *метка 1, метка 2, метка 3, ...*

Все необходимые модификации компонуемых двоичных модулей производятся компоновщиком автоматически, без какого-либо вмешательства программиста. Используя специальные приемы программирования, можно составлять программы, не требующие модификации при загрузке в разные участки памяти. Совокупность таких приемов называется *позиционно-независимым кодированием.*

Отладка программы

Отладка — это процесс обнаружения и устранения ошибок в программе. Лишь в редких случаях вновь написанная программа сразу же начинает работать. Но это еще не означает, что она написана правильно: ошибка может проявиться позднее при каком-то определенном сочетании входных данных (скрытая ошибка). Зная это, многие опытные программисты исходят из правила: «Любая программа содержит ошибки». Можно только порадоваться вместе с читателем, которому посчастливится неоднократно опровергнуть это утверждение, но чаще всего избежать ошибок удается лишь в весьма простых программах. И дело не только в том, что программисту нужно определенное время, чтобы научиться правильно выражать свои мысли (точнее — алгоритмы)

на языке, все-таки далеком от естественного. Основная причина ошибок, допускаемых даже опытными программистами, состоит в невозможности предсказать поведение сложной разветвленной программы во всех мыслимых ситуациях и при любых наборах исходных данных. Такие ошибки — настоящий бич программирования, а цена их может быть очень велика. Известны случаи, когда в результате единственного неверного оператора срывались запуски космических кораблей, происходили серьезные аварии на производстве и транспорте. Поэтому разработка методов, облегчающих процесс обнаружения ошибок и позволяющих проверять правильность программ,— это одна из центральных проблем информатики.

Существует ряд рекомендаций, снижающих вероятность появления ошибок в программах:

— использовать по мере возможности языки высокого уровня;

— разбивать программу на отдельные законченные модули, предусмотрев возможность индивидуальной отладки каждого модуля;

— широко использовать готовые отлаженные подпрограммы из библиотек;

— избегать запутанных, излишних ветвлений, различных «трюков», призванных, например, сэкономить несколько слов в ОЗУ;

— использовать ясные и точные комментарии в программе.

Все ошибки, встречающиеся в программах, подразделяются на логические, синтаксические и опечатки, причем лишь синтаксические ошибки и в некоторых случаях опечатки могут быть обнаружены автоматически программой-транслятором. Наибольшие трудности представляет обнаружение логических ошибок, для чего существует целый ряд методов. Например, в текст программы можно временно включить специальные отладочные операторы или подпрограммы, позволяющие вывести на печать промежуточные результаты в критических точках выполнения программы, а по окончании отладки удалить их.

Как правило, каждый транслятор имеет какие-либо отладочные средства. Так, для отладки программ на языке Фортран существует специальный отладочный объектный модуль FDT (Fortran Debugging Technigue), подключаемый на этапе компоновки к отлаживаемой программе. При работе с Бейсиком для отладки используют точки останова и операторы, выполняемые в непосредственном режиме, а для отладки ассемблерных программ существует несколько модулей-отладчиков, из которых «классическим» является ODT (On-line Debugging Technique).

Вместо заключения: достижения и перспективы микропроцессорной техники

«Если бы за последние 25 лет авиационная промышленность развивалась столь же стремительно, как и вычислительная тех-

ника, то Боинг-767 можно было бы приобрести за 600 долл. и облететь на нем земной шар за 20 мин, израсходовав при этом 19 л горючего»,— эта цитата из журнала «В мире науки» как нельзя лучше отражает беспрецедентные темпы развития вычислительной техники, прежде всего — микропроцессорной. Действительно, с 1971 г. (год выпуска первого микропроцессора — Intel 4004) при снижении цен на освоенные промышленностью микропроцессоры в десятки и сотни раз количество транзисторов на кристалле возросло примерно в 200 раз, а производительность — в сотни и даже тысячи раз. Поэтому современный микропроцессорный персональный компьютер обладает вычислительной мощностью большей, чем огромный вычислительный центр 50-х годов при энергопотреблении меньшем в десятки тысяч раз.

За время создания этой книги на смену рассмотренному в ней микропроцессору К1801ВМ1 пришли его более мощные «собратья» — ВМ2 и ВМ3. О возможностях последнего можно судить хотя бы по размеру адресуемой памяти — 1920 Кслов. Эти МП — основа новых ЭВМ ДВК-3 и ДВК-4, немаловажное достоинство которых — наличие развитых средств для работы с графической информацией.

Вкратце рассмотрим другие важные достижения микропроцессорной техники и перспективы ее развития. Своеобразным «законодателем мод» остается фирма Intel (США). Восьмиразрядный МП Intel 8080, появившийся еще в 1974 г., широко применяется по сей день, а его усовершенствованные варианты (Intel 8085, Motorola 6800, Mostek 6502, Zilog 80) — основа массовых восьмиразрядных компьютеров, таких, как MSX фирмы Yamaha, ZX — Spectrum фирмы Sinclair, Apple-II, «Агат», «Ириша», «Микроша», «Корвет», Robotron-1715 и других.

Не меньший успех среди шестнадцатиразрядных МП имеют модели Intel 8086, 8088, 80186, 80286, в которых воплощено множество интересных идей. Например, они содержат небольшую встроенную память для команд, организованную по типу очереди. В то время как операционный блок выполняет текущую операцию, блок связи с магистралью автоматически извлекает из ОЗУ следующую команду и ставит ее в очередь на выполнение. Такой «конвейерный» принцип считывания и выполнения команд ощутимо повышает производительность МП. Еще одна особенность — возможность подключения отдельных БИС математических сопроцессоров, позволяющих в несколько раз повысить скорость вычислений. На МП этого семейства создаются самые популярные на сегодняшний день персональные компьютеры IBM PC и их отечественные аналоги — ЕС 1840, ЕС 1841, «Искра-1030», «Нейрон».

Одно из наиболее впечатляющих достижений — создание 32-разрядного МП Intel 80386. Уже сейчас на его основе налажен выпуск «старших» моделей нового семейства персональных компьютеров фирмы IBM — так называемых персональных систем PS/2. Разработчики утверждают, что своим вычислительным возможностям МП 80386 и первый МП 4004 соотносятся друг с другом

приблизительно так же, как космический корабль «Шаттл» и аэроплан братьев Райт.

А что же последует за 32-разрядными микропроцессорами? Специалисты склонны считать, что напрашивающегося перехода к 64-разрядным МП не произойдет. Действительно, «большие» ЭВМ уже более 40 лет остановились на 32 разрядах и похоже, что такой длины слова вполне хватает для большинства задач. Можно ожидать появления 64-разрядных шин между процессором и памятью, но по существующим оценкам развитие МП скорее всего пойдет по путям совершенствования их архитектуры. Одно из направлений — создание МП с сокращенным набором команд, или процессоров типа RISC (Reduced Instruction Set Computer). Сторонники этого направления расценивают усложнение обычных МП как чрезмерное, поскольку из сотен команд, которые «умеет» выполнять типичный современный МП, в среднем сколь-либо интенсивно используется лишь небольшая их часть. Таким образом, 80—90 % аппаратуры МП в среднем бездействует. Кстати, самая первая массовая мини-ЭВМ PDP-8, созданная еще в 60-х годах, имела лишь 8 основных команд. Типичный RISC-процессор рассчитан лишь на десятки команд, каждая из которых выполняется за один машинный цикл, а не за несколько, как у обычных МП. Выпускаемые в настоящее время десятки различных моделей 16- и 32-разрядных RISC-МП отличаются повышенным быстродействием (до 100 млн операций в с), высокой технологичностью изготовления, а следовательно, высокой надежностью и меньшей стоимостью.

Другое направление привело к созданию уникальных приборов, получивших название «транспьютер». Транспьютер — это однокристальная ЭВМ, содержащая процессор (обычно RISC), память, интерфейсы внешней памяти и периферии, а также средства, обеспечивающие многозадачный режим работы и взаимодействие нескольких транспьютеров. В числе этих средств — скоростные последовательные линии связи (обычно — 4), которые позволяют объединять множество транспьютеров в ячеистые структуры с параллельной обработкой данных. Потенциальные возможности структур из множества транспьютеров трудно переоценить, причем препятствием на пути их широкого распространения является не техническая реализация, а разработка математического обеспечения параллельной обработки данных.

Транспьютерный суперкомпьютер — уже факт. На очереди — реализация новых архитектур из транспьютеров. Может быть, за ними — будущее вычислительной техники. Однако не будем торопиться с прогнозами, поскольку практика нередко перечеркивает даже самые правдоподобные из них.

МАЛЫЕ ЭВМ

СПРАВОЧНИК

ПО
СИСТЕМЕ
КОМАНД

Электроника-60 ★ Электроника-МС 1201 ★ СМ3 ★ СМ 1300 ★ Электроника-100/25 ★ Электроника-МС 1211 ★ Электроника-МС 1212 ★ СМ4 ★ СМ 1600 ★ СМ 1420 ★ Электроника-79

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ

В КОДАХ МАШИННЫХ КОМАНД

■ = 0 для операций над словами
1 для операций над байтами
SS = поле адресации операнда-источника
DD = поле адресации операнда-приемника
R = регистр общего назначения № 0...7 (3 бита)
XXX = смещение (−128..., +127, 8 бит)
N = число, 3 бита
NN = число, 6 бит

В ОБОЗНАЧЕНИЯХ ОПЕРАЦИЙ

( ) = содержимое
s = операнд-источник
d = операнд-приемник
r = содержимое регистра
← = становится равным
X = относительный адрес
% = определение регистра

ЛОГИЧЕСКИХ ОПЕРАЦИЙ

∧ = И
∨ = ИЛИ
∀ = исключающее ИЛИ
~ = НЕ

ОПЕРАЦИЙ НАД РАЗРЯДАМИ ССП

* = установка/сброс в зависимости от результата
— = состояние разряда не изменяется
0 = сброс
1 = установка

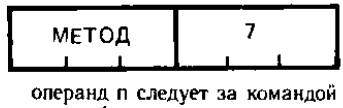
РАСШИРЕННЫЕ НАБОРЫ КОМАНД:

● — EIS ▲ — FIS ▼ — FPP

МЕТОДЫ АДРЕСАЦИИ

Метод	Наименование	Мнемоника	Описание
0	регистровая	R	(R) — операнд
1	косвенная регистровая	(R) или @ R	(R) — адрес операнда
2	автоинкрементная	(R)+	(R) — адрес (R) ← (R) + 1 (или 2)
3	косвенная автоинкрементная	@(R)+	(R) — адрес адреса, (R) ← (R) + 1 (или 2)
4	автодекрементная	−(R)	(R) ← (R) − 1 (или 2) (R) — адрес
5	косвенная автодекрементная	@ −(R)	(R) ← (R) − 2, (R) — адрес адреса
6	индексная	X(R)	(R) + X — адрес
7	косвенная индексная	@X(R)	(R) + X — адрес адреса

С ИСПОЛЬЗОВАНИЕМ СЧЕТЧИКА КОМАНД

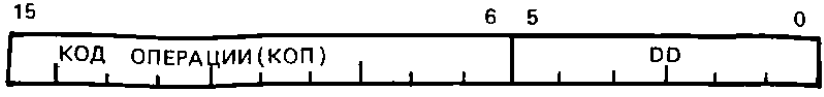


2	непосредственная	#n	операнд n следует за командой
3	абсолютная	@#A	адрес A следует за командой
6	относительная	A	адрес операнда = адрес команды + 4 + X
7	косвенная относительная	@A	адрес адреса операнда = адрес команды + 4 + X

ВЕКТОРЫ ПРЕРЫВАНИЙ

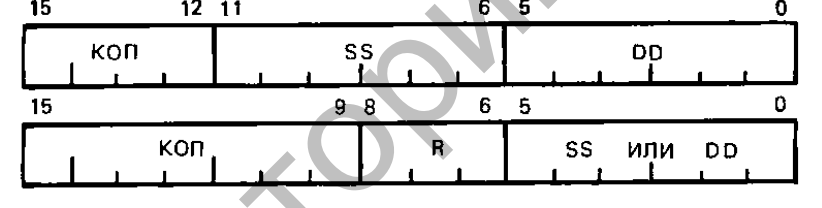
000 — резерв
004 — тайм-аут магистрали и др. ошибки
010 — попытка исполнения недопустимой или резервной команды
114 — контроль четности
014 — команда BPT
020 — команда IOT
024 — авария питания
030 — команда EMT
034 — команда TRAP
244 — FPP
250 — прерывание диспетчера ОЗУ

ОДНООПЕРАНДНЫЕ КОМАНДЫ: OPR dst



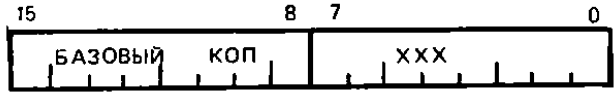
Мнемоника	КОП	Наименование	Операция	N	Z	V	C
		КОМАНДЫ ОБЩЕГО НАЗНАЧЕНИЯ					
CLR(B)	050DD	очистка	d←0	0	1	0	0
COM(B)	051DD	побитная инверсия	d←~d	*	*	0	1
INC(B)	052DD	прибавление единицы	d←d+1	*	*	*	—
DEC(B)	053DD	вычитание единицы	d←d−1	*	*	*	—
NEG(B)	054DD	изменение знака	d←−d	*	*	*	*
TST(B)	057DD	проверка	d←d	*	*	0	0
		СДВИГИ					
ROR(B)	060DD	циклич. сдвиг вправо	→C, d	*	*	*	*
ROL(B)	061DD	циклич. сдвиг влево	C, d←	*	*	*	*
ASR(B)	062DD	арифм. сдвиг вправо	d←d/2	*	*	*	*
ASL(B)	063DD	арифм. сдвиг влево	d←2d	*	*	*	*
SWAB	0003DD	перестановка байтов		*	*	0	0
		ОПЕРАЦИИ ПОВЫШЕННОЙ ТОЧНОСТИ					
ADC(B)	055DD	прибавление переноса	d←d+C	*	*	*	*
SBC(B)	056DD	вычитание переноса	d←d−C	*	*	*	*
● SXT	0067DD	расширение знака	d←0 или −1	—	*	0	—

ДВУХОПЕРАНДНЫЕ КОМАНДЫ: OPR src, dst; OPR src, R; OPR R, dst



Мнемоника	КОП	Наименование	Операция	N	Z	V	C
		КОМАНДЫ ОБЩЕГО НАЗНАЧЕНИЯ					
MOV(B)	1SSDD	пересылка	d←s	*	*	0	—
CMP(B)	2SSDD	сравнение	s−d	*	*	*	*
ADD	06SSDD	сложение	d←s+d	*	*	*	*
SUB	16SSDD	вычитание	d←d−s	*	*	*	*
		ЛОГИЧЕСКИЕ КОМАНДЫ					
BIT(B)	3SSDD	поразрядная проверка	s∧d	*	*	0	—
BIC(B)	4SSDD	поразрядная очистка	d←(~s)∧d	*	*	0	—
BIS(B)	5SSDD	поразрядная установка	d←s∨d	*	*	0	—
		РЕГИСТРОВЫЕ КОМАНДЫ					
● MUL	070RSS	умножение	r←r×s	*	*	0	*
● DIV	071RSS	деление	r←r/s	*	*	*	*
● ASH	072RSS	арифметический сдвиг		*	*	*	*
● ASHC	073RSS	арифм. сдвиг двойного слова		*	*	*	*
	074RSS	исключающее ИЛИ	d←r∀d	*	*	0	—

ПЕРЕХОДЫ ПО УСЛОВИЮ (ВЕТВЛЕНИЯ): B** <адрес перехода>
Если условие удовлетворяется, то (СК) ← (СК) + (2 × XXX)



Мнемоника	Базовый КОП	Наименование условия ветвления	Условие ветвления
BR	000400	безусловное ветвление	—
BNE	001000	нет равенства (нулю)	Z = 0
BEQ	001400	равенство (нулю)	Z = 1
BPL	100000	знак плюс	N = 0
BMI	100400	знак минус	N = 1
BVC	102000	арифм. переполнение отсутствует	V = 0
BVS	102400	произошло арифм. переполнение	V = 1
BCC	103000	перенос отсутствует	C = 0
BCS	103400	произошел перенос	C = 1
		ЗНАКОВЫЕ ВЕТВЛЕНИЯ	
BGE	002000	больше или равно (нулю)	N ∀ V = 0
BLT	002400	меньше (нуля)	N ∀ V = 1
BGT	003000	больше (нуля)	Z ∨ (N ∀ V) = 0
BLE	003400	меньше или равно (нулю)	Z ∨ (N ∀ V) = 1
		БЕЗЗНАКОВЫЕ ВЕТВЛЕНИЯ	
BHI	101000	больше	C ∨ Z = 0
BLOS	101400	меньше или равно	C ∨ Z = 1
BHIS	103000	больше или равно	C = 0
BLO	103400	меньше	C = 1

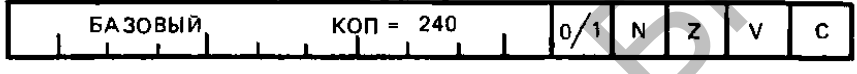
ПЕРЕХОД, РАБОТА С ПОДПРОГРАММАМИ И ПРЕРЫВАНИЯМИ

Мнемоника	КОП	Наименование	Операция
JMP	0001DD	безусловный переход	СК ← d
JSR	004RDD	вызов подпрограммы	— (R6) ← R ← СК ← d
RTS	00020R	возврат из подпрограммы	СК ← R ← (R6) +
● MARK	0064NN	восстановление стека	R6 ← СК + 2 × NN, СК ← ← R5 ← (R6) +
● SOB	077RNN	вычитание 1 и ветвление, если ≠ 0	R ← R — 1, если (R) = 0, то СК ← ← СК — (2 × NN)
		ПРОГРАММНЫЕ ПРЕРЫВАНИЯ	
EMT	104000—104377	вызов ОС, вектор 30	— (R6) ← ССП ← (32), — (R6) ← СК ← (30)
TRAP	104400—104777	общего назначения, вектор 34	— (R6) ← ССП ← (36), — (R6) ← СК ← 34
BPT	000003	отладочное, вектор 14	— (R6) ← ССП ← 16, — (R6) ← СК ← 14
IOT	000004	вызов системы ввода-вывода, вектор 20	— (R6) ← ССП ← 22, — (R6) ← СК ← 20
		ВОЗВРАТ ИЗ ПРЕРЫВАНИЙ	
RTI	000002	возврат из прерывания	СК ← (R6) +, ССП ← (R6) +
● RTT	000006	то же, с запретом прерывания по Т-разряду до исполнения следующей команды	СК ← (R6) +, ССП ← (R6) +

РАЗНЫЕ:

Мнемоника	КОП	Наименование	Операция
HALT	000000	останов	—
WAIT	000001	пауза — ожидание прерывания	—
RESET	000005	сброс магистрали и ЦП	—
NOP	000240	нет операции (пустая команда)	—
● SPL	00023N	установить уровень приоритета	ССП (5...7) ← N
● MFPI	0065SS	пересылка из области инструкций пред. режима	— (R6) ← s
● MTPI	0066DD	пересылка в область инструкций пред. режима	d ← (R6) +
● MFPD	1065SS	пересылка из области данных пред. режима	— (R6) ← s
● MTPD	1066DD	пересылка в область данных пред. режима	d ← (R6) +
MFPS	1067DD	чтение ССП	d ← ССП
MTPS	1064SS	запись ССП	ССП ← s

ОПЕРАЦИИ С РАЗРЯДАМИ ССП:

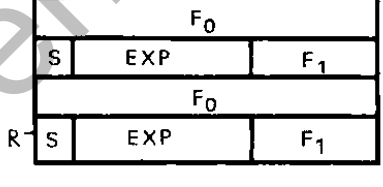


0 — очистить отмеченные разряды ССП
1 — установить

Мнемоника	КОП	Наименование	N	Z	V	C
CLC	000241	очистка C	—	—	—	0
CLV	000242	очистка V	—	—	0	—
CLZ	000244	очистка Z	—	0	—	—
CLN	000250	очистка N	0	—	—	—
CCC	000257	очистка C, V, Z, N	0	0	0	0
SEC	000261	установка C	—	—	—	1
SEV	000262	установка V	—	—	1	—
SEZ	000264	установка Z	—	1	—	—
SEN	000270	установка N	1	—	—	—
SCC	000277	установка C, V, Z, N	1	1	1	1

▲ СТЕКОВЫЕ ОПЕРАЦИИ С ПЛАВАЮЩЕЙ ТОЧКОЙ:

Мнемоника	КОП	Наименование	Операция	N	Z	V	C
FADD	07500R	сложение	B ← A + B	*	*	0	0
FSUB	07501R	вычитание	A ← A — B	*	*	0	0
FMUL	07502R	умножение	B ← A * B	*	*	0	0
FDIV	07503R	деление	B ← B/A	*	*	0	0



S — знак числа (1 бит)
EXP — порядок в коде со смещением 128_{10} (8 бит)
F_0 — младшая часть мантиссы (16 бит)
F_1 — старшая часть мантиссы (7 бит)

▼ КОМАНДЫ ПРОЦЕССОРА С ПЛАВАЮЩЕЙ ТОЧКОЙ (FPP):

Мнемоника	КОП	Наименование	Операция
CFCC	170000	чтение ССП FPP в ССП ЦП	C, V ← FC, FV Z, N FZ, FN
SETF	170001	установить режим плавающей точки	FD ← 0
SETI	170002	установить режим целых чисел	FL ← 0
SETD	170011	установить режим двойной точности	FD ← 1
SETL	170012	установить режим «длинных целых»	FL ← 1
LDFPS	1701SS	запись регистра статуса FPP	FPS ← s
STEPS	1702DD	чтение регистра статуса FPP	d ← FPS
STST	1703DD	чтение состояния FPP	d ← (FEC), d + 2 ← (FEA)
CLRF, CLRD	1704DD	очистка	d ← 0
TSTF, TSTD	1705DD	проверка	d ← d
ABSF, ABSD	1706DD	модуль	d ← + / — d
NEGF, NEGD	1707DD	изменение знака	d ← — d
MULF, MULD	1710 AC SS	умножение	AC ← AC*s
MODF, MODD	1714 AC SS	умножение с выделением целой и дробной частей	
ADDF, ADDD	1720 AC SS	сложение	AC ← AC + s
LDF, LDD	1724 AC SS	загрузка аккумулятора	AC ← s
SUBF, SUBD	1730 AC SS	вычитание	AC ← AC — s
CMPF, CMPD	1734 AC SS	сравнение	s — AC
STF, STD	1740 AC DD	чтение аккумулятора	d ← AC
DIVF, DIVD	1744 AC SS	деление	AC ← AC/s
STEXP	1750 AC DD	чтение порядка числа, записанного в AC	d ← EXP(AC) — — 200
STCFI, STCFL STCDI, STCDL	1754 AC DD	чтение с преобразованием форматов плав. чисел в целые	
STCFD, STCDF	1760 AC DD	чтение с преобр. числа двойной длины	
LDEXP	1764 AC SS	загрузка порядка числа в AC	EXP(AC) — d + + 200
LDCIF, LDCID LDCLF, LDCLD	1770 AC SS	загрузка с преобразованием целых форматов в плавающие	
LDCDF, LDCFD	1774 AC SS	загрузка с преобразованием плав. форматов	

ТАБЛИЦА НАЧАЛЬНОЙ ЗАГРУЗКИ ЭВМ

Стартовый адрес загрузчика/ команда монитора	Устройство загрузки	Примечания
1		
2		
3		
4		

КОДЫ МАШИННЫХ КОМАНД:

КОП	Мнемоника
00 00 00	HALT
00 00 01	WAIT
00 00 02	RTI
00 00 03	BPT
00 00 04	IOT
00 00 05	RESET
● 00 00 06	RTT
00 00 07	
...	резерв
00 00 77	
00 01 DD	JMP
00 02 0R	RTS
00 02 10	
...	резерв
00 02 27	
● 00 02 3N	SPL
00 02 40	NOP
00 02 41	операции с
...	разрядами
00 02 77	ССП
00 03 DD	SWAB
00 04 XXX	BR
00 10 XXX	BNE
00 14 XXX	BEQ
00 20 XXX	BGE
00 24 XXX	BLT
00 30 XXX	BGT
00 34 XXX	BLE
00 4R DD	JSR
00 50 DD	CLR
00 51 DD	COM
00 52 DD	INC
00 53 DD	DEC
00 54 DD	NEG
00 55 DD	ADC
00 56 DD	SBC
00 57 DD	TST

КОП	Мнемоника
00 60 DD	ROR
00 61 DD	ROL
00 62 DD	ASR
00 63 DD	ASL
● 00 64 NN	MARK
00 65 SS	MFPI
00 66 DD	MTPI
● 00 67 DD	SXT
00 70 00	
...	резерв
00 77 77	
01 SS DD	MOV
02 SS DD	CMP
03 SS DD	BIT
04 SS DD	BIC
05 SS DD	BIS
06 SS DD	ADD
● 07 0R SS	MUL
● 07 1R SS	DIV
● 07 2R SS	ASH
● 07 3R SS	ASHC
● 07 4R SS	XOR
▲ 07 50 0R	FADD
▲ 07 50 1R	FSUB
▲ 07 50 2R	FMUL
▲ 07 50 3R	FDIV
07 50 40	
...	резерв
07 67 77	
● 07 7R NN	SOB
10 00 XXX	BPL
10 04 XXX	BMI
10 10 XXX	BHI
10 14 XXX	BLOS
10 20 XXX	BVC
10 24 XXX	BVS
10 30 XXX	BCC, BHIS
10 34 XXX	BCS, BLO

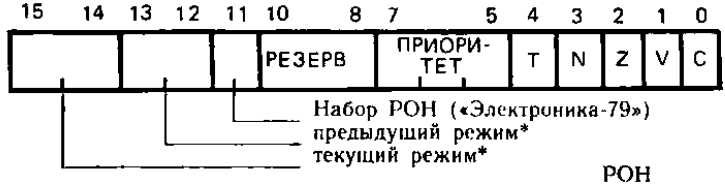
КОП	Мнемоника
10 40 00	
...	EMT
10 43 77	
10 44 00	
...	TRAP
10 47 77	
10 50 DD	CLRB
10 51 DD	COMB
10 52 DD	INCB
10 53 DD	DECB
10 54 DD	NEGB
10 55 DD	ADCB
10 56 DD	SBCB
10 57 DD	TSTB
10 60 DD	RORB
10 61 DD	ROLB
10 62 DD	ASRB
10 63 DD	ASLB
10 64 SS	MTPS
10 65 SS	MFPD
10 66 DD	MTPD
10 67 DD	MFPS
10 70 00	
...	резерв
10 77 77	
11 SS DD	MOVB
12 SS DD	CMPB
13 SS DD	BITB
14 SS DD	BICB
15 SS DD	BISB
16 SS DD	SUB
▼ 17 00 00	
	FPP
▼ 17 77 77	

Примечание:

MTPS, MFPS — для процессоров с интерфейсом типа МПИ
SPL — для процессоров СМ1420, СМ1600, Электроника-79
MFPI, MTPI — для процессоров с MMU
MFPD, MTPD — для процессоров с 22-разрядным MMU
SOB, SXT, MARK
RTT, XOR — выполняются в МП К1801ВМ1

АДРЕСА РЕГИСТРОВ ПРОЦЕССОРА:

Слово состояния процессора (ССП) — X77776 X = 1 — 16 разр. ЭВМ
X = 7 — 18 разр. ЭВМ
X = 177 — 22 разр. ЭВМ



*00 = KERNEL, 01 = SUPERVISOR, 11 = USER

Регистр — ограничитель стека — X77774
Регистр запросов прерываний — X77772
Клавишный регистр — X77570
Статусный регистр сетевого таймера — X77546

РОН

	Набор 0	Набор 1
R0	X77700	X77710
R1	X77701	X77711
R2	X77702	X77712
R3	X77703	X77713
R4	X77704	X77714
R5	X77705	X77715
R6	X77706(K), X77717(U)	X77716(S),
R7(CK)	X77707	

Диспетчер памяти (MMU):

Режимы ЦП	Область пам.	Адреса регистров: адресов страниц (PAR)	описания страниц (PDR)
KERNEL	I	X72340—072356	X72300 — X72316
	D	X72360 — X72376	X72320 — X72336
USER	I	X77640 — X77656	X77600 — X77616
	D	X77660 — X77676	X77620 — X77636
SUPERVISOR	I	X72240 — X72256	X77200 — X77216
	D	X72260 — X72276	X77220 — X77236

Статусные регистры: SR0 — X77572, SR1 — X77574, SR2 — X77576, SR3 — X72516

Регистры управления адресацией магистрали «Общая Шина» — X70200 — X70366

КОМАНДЫ ПУЛЬТОВОГО ЭМУЛЯТОРА (ДЛЯ МИКРО-ЭВМ):

/ — открыть ячейку
↑ — открыть предыдущую ячейку
<ПС> — открыть следующую ячейку
@ — открыть ячейку с абсолютным адресом
— открыть ячейку с относительным адресом
<ВК> — закрыть ячейку
G — пуск программы
P — продолжение программы
<ЗБ> — удаление ошибочного символа
M — причина останова
RS — обращение к ССП
R_n — обращение к РОН ($n = 0 \ldots 7$)
> — открыть ячейку по адресу перехода
T0 — пуск цепочки тестов
T_n — пуск отдельных тестов ($n = 1 \ldots 6$)
} для МС 1201.01, 02
B — вызов монитора начальной загрузки (для МС 1201.02)
X*n* — загрузка ОС с НГМД-6022 ($n = 0 \ldots 3$)
D*n* — загрузка ОС с ГМД-7012, ГМД-70
XL — загрузка с перфоленты, X — адрес регистра состояния считывателя

АДРЕСА РЕГИСТРОВ ВНЕШНИХ УСТРОЙСТВ:

		Адрес	Вектор	Приоритет
Консольный видеотерминал (№ 0):				
состояния клавиатуры	KBS	X77560	60	4
данных клавиатуры	KBD	X77562		
состояния экрана	TPS	X77564	64	4
данных экрана	TPB	X77566		
Регистры состояния клавиатуры дополнительных терминалов:				
	№ 1	X76500	300	4
	№ 2	X76510	310	4
	№ 3	X76520	320	4
	№ 4	X76530	330	4
	№ 5	X76540	340	4
	№ 6	X76550	350	4
	№ 7	X76560	360	4
Устройство печати:				
состояния	LPCS	X77514	200	4
данных	LPDB	X77516		
НГМД (ГМД-70, ГМД-7012, RX-01, RX-02):				
управления/состояния	RXCS	X77170	264	5
данных	RXDB	X77172		
Кассетный НМЛ (СМ5211, TA11):				
управления/состояния	TACS	X77500	260	6
данных	TADB	X77502		
Комбинированное перфоленточное устройство:				
состояния считывателя	PRS	X77550	70	4
данных считывателя	PRB	X77552		
состояния перфоратора	PPS	X77554	74	4
данных перфоратора	PPB	X77556		
НМД 2,5 М байт (СМ5400, RK05):				
статус привода	RKDS	X77400	220	5
ошибки	RKER	X77402		
управления/состояния	RKCS	X77404		
счетчик слов	RKWC	X77410		
текущий адрес	RKBA	X77412		
адрес диска	RKDA	X77414		
буфер данных	RKDB	X77416		
НМЛ (СМ5300, TMA11/TU10):				
состояния	MTS	X72520	224	5
управления	MTC	X72522		
счетчик байтов записи	MTBRC	X72524		
текущий адрес	MTCMA	X72526		
буфер данных	MTD	X72530		
чтения строк	MTRD	X72532		

АЛФАВИТНО-ЦИФРОВОЙ КОД ASCII и КОИ-7

Код$_8$	Символ	Код$_8$	Символ	Код$_8$	Символ	Код$_8$	ASCII	КОИ 7
000	NUL	040	SP (пробел)	100	@	140	`	Ю
001	SOH	041	!	101	A	141	a	А
002	STX	042	«	102	B	142	b	Б
003	ETX	043	#	103	C	143	c	Ц
004	EOT	044	$ (¤)	104	D	144	d	Д
005	ENQ	045	%	105	E	145	e	Е
006	ACK	046	&	106	F	146	f	Ф
007	BEL	047	'	107	G	147	g	Г
010	BS	050	(	110	H	150	h	Х
011	HT	051	)	111	I	151	i	И
012	LF	052	*	112	J	152	j	Й
013	VT	053	+	113	K	153	k	К
014	FF	054	,	114	L	154	l	Л
015	CR	055	—	115	M	155	m	М
016	SO	056	.	116	N	156	n	Н
017	SI	057	/	117	O	157	o	О
020	DLE	060	0	120	P	160	p	П
021	DC1	061	1	121	Q	161	q	Я
022	DC2	062	2	122	R	162	r	Р
023	DC3	063	3	123	S	163	s	С
024	DC4	064	4	124	T	164	t	Т
025	NAK	065	5	125	U	165	u	У
026	SYN	066	6	126	V	166	v	Ж
027	ETB	067	7	127	W	167	w	В
030	CAN	070	8	130	X	170	x	Ь
031	EM	071	9	131	Y	171	y	Ы
032	SUB	072	:	132	Z	172	z	З
033	ESC	073	;	133	[	173	{	Ш
034	FS	074	<	134	\	174	|	Э
035	CS	075	=	135	]	175	}	Щ
036	RS	076	>	136	↑	176	~	Ч
037	US	077	?	137	—	177	Забой	

ТАБЛИЦА СПЕЦСИМВОЛОВ ДЛЯ ВИДЕОТЕРМИНАЛА «Электроника 15ИЭ-00-013»

Функция	Режим I	Режим II	Функция	Режим I	Режим II
Курсор вверх	СУ/\	ESC, A	Размыкание строки	СУ/S	—
Курсор вниз	СУ/]	ESC, B	Смыкание строки	СУ/T	—
Курсор влево	СУ/Z	ESC, D	Передача экрана	СУ/↑	—
Курсор вправо	СУ/Y	ESC, C	Передача строки	СУ/F	—
Курсор в начало экрана	СУ/H	ESC, H	Очистка экрана	СУ/L	—
Звуковой сигнал	СУ/G	СУ/G	Очистка экрана от курсора	СУ/—	ESC, J
Перевод строки	СУ/J	СУ/J	Очистка строки от курсора	СУ/K	ESC, K
Возврат каретки	СУ/M	СУ/M	Автоответ	—	ESC, Z
Латинский набор символов	СУ/O	СУ/O	Установка курсора в начало следующей строки	СУ/U	—
Русский набор символов	СУ/N	СУ/N	Прямая адресация курсора	—	ESC, Y, $y+40$, $x+40$
Сдвиг текста вверх	СУ/R	—			
Сдвиг текста вниз	СУ/V	—			
Переход в режим I	—	ESC, E			
Переход в режим II	СУ/W	—			
Установка альтерн. режима	—	ESC, =			
Снятие альтерн. режима	—	ESC, >			

ОГЛАВЛЕНИЕ

ГЛАВА 4

ГЛАВА 5

ГЛАВА 6

ГЛАВА 7

ГЛАВА 8

Научно-популярное издание

Бильдюкевич Евгений Викторович
Гурачевский Валерий Леонидович
Шушкевич Станислав Станиславович

ЭВМ И МИКРОПРОЦЕССОР

Книга для учащихся

Заведующий редакцией *Б. А. Кимбар*
Редактор *В. В. Амбражевич*
Оформление художника *А. А. Богуш*
Исполнительная графика *В. Ф. Чекмарева*
Художественный редактор *Н. И. Евменова*
Технический редактор *З. В. Романкевич*
Корректоры *Р. С. Ахремчик, И. С. Еремчик*

ИБ № 2188

Сдано в набор 25.02.88. Подписано в печать 09.06.89. АТ 10727. Формат 60 × 90$^{1}/_{16}$. Бумага кн.-журнальная. Гарнитура литературная. Офсетная печать. Усл. печ. л. 13. Усл. кр.-отт. 27. Уч.-изд. л. 12,71. Тираж 44 000 экз. Заказ 433. Цена 85 к.

Издательство «Народная асвета» Государственного комитета БССР по печати. 220600, Минск, проспект Машерова, 11.

Набрэко на Минском ордена Трудового Красного Знамени полиграфкомбинате МППО имени Я. Коласа. 220005, Минск, Красная, 23.

Минская фабрика цветной печати. 220115, Минск, Корженевского, 20.

СК	**счетчик команд**
СОЗУ	**сверхоперативное запоминающее устройство**
ССП	**слово состояния процессора**
СТР	**строб (сигнал ИРПР)**
СУ	**символ управления**
СУБД	**система управления базами данных**
ТТЛ	**транзисторно-транзисторная логика**
УВВ	**устройство ввода-вывода информации**
УВХ	**устройство выборки и хранения**
УМТ	**универсальная машина Тьюринга**
УС	**указатель стека**
УУ	**устройство управления**
ФОДОС	**фоново-оперативная дисковая операционная система**
ЦА	**цифровой автомат**
ЦАП	**цифро-аналоговый преобразователь**
ЦП	**центральный процессор**
Э	**эмиттер (вывод транзистора)**
ЭВМ	**электронная вычислительная машина**
ЭЛТ	**электронно-лучевая трубка**
ЭС	**экспертная система**
ЭСЛ	**эмиттерно-связанная логика**
ASCII	**(American Standard Code for Information Interchange) — американский стандартный код обмена информацией**
BASIC	**(Beginner's All-purpose Symbolic Instruction Code) — универсальный символический код для начинающих (язык Бейсик)**
C	**(Carry) — перенос (бит переноса в ССП)**
C	**(Clock) — тактовый сигнал**
D	**(Delay) — триггер задержки, D — триггер**
DEC	**(Digital Equipment Corp.) — торговый знак крупнейшего в США производителя малых ЭВМ**
FIFO	**(First In — First Out) — организация памяти по принципу «первым вошел — первым вышел» (очередь)**

FORTRAN	(FORmula TRANslation) — перевод формул — название языка программирования высокого уровня (язык Фортран)
JK	(Jump-Keep) — прыгай-держись (разновидность триггера)
LIFO	(Last In-First Out) — организация памяти по принципу «последним вошел — первым вышел» (стек)
LSI	(Large Scale Integration) — высокая степень интеграции
MMU	(Memory Management Unit) — устройство управления памятью, диспетчер памяти
MS	(Master-Slave) — хозяин-раб (разновидность триггера)
N	(Negative) — признак отрицательного результата в ССП
n	полупроводник с электронной проводимостью (n-типа)
p	полупроводник с дырочной проводимостью (p-типа)
PDP	(Programmed Data Processor) — программируемый процессор данных (торговый знак семейства мини-ЭВМ фирмы DEC)
T	(Trace) — бит трассировки в ССП
R	(Register) — регистр
RAM	(Random Access Memory) — запоминающее устройство с произвольной выборкой
ROM	(Read-Only Memory) — постоянное запоминающее устройство
RS	(Reset-Set) — сброс-установка (тип триггера)
RSX	(Resource Sharing eXecutive) — диспетчер распределения ресурсов (торговый знак операционной системы фирмы DEC)
RT	(Real Time) — реальное время
S	(Set) — установка
V	(oVerflow) — бит переполнения в ССП
Z	(Zero) — признак нулевого результата в ССП

СПИСОК СОКРАЩЕНИЙ

АЛУ	**арифметико-логическое устройство**
АЦП	**аналого-цифровой преобразователь**
Б	**база (вывод транзистора)**
БД	**база данных**
БИС	**большие интегральные схемы**
ВЗУ	**внешнее запоминающее устройство**
ВК	**возврат каретки (символ алфавитно-цифрового кода)**
ВС	**вычислительная система**
ВУ	**внешнее устройство**
ГП	**готовность приемника (сигнал ИРПР)**
ГПС	**гибкая производственная система**
ДВК	**диалоговый вычислительный комплекс**
ДТЛ	**диодно-транзисторная логика**
ЕС	**Единое Семейство ЭВМ**
З	**затвор (вывод транзистора)**
ЗБ	**забой (символ алфавитно-цифрового кода)**
ЗП	**запрос приемника (сигнал ИРПР)**
ЗУ	**запоминающее устройство**
ЗУПВ	**запоминающее устройство с произвольной выборкой**
ЗЭ	**запоминающий элемент**
И	**исток (вывод транзистора)**
$И^2Л$, ИИЛ	**интегральная инжекционная логика**
ИПС	**информационно-поисковая система**
ИРПР	**интерфейс радиальный параллельный**
ИРПС	**интерфейс радиальный последовательный**
ИС	**интегральная схема**
К	**коллектор (вывод транзистора)**
КМОП	**МОП — технология с комплементарными транзисторами**
КОИ-7	**код обмена информацией, семибитный**
КОИ-8	**код обмена информацией, восьмибитный**
КОП	**код операции**
КС	**комбинационная схема**
МДП	**металл-диэлектрик-полупроводник**

МОП	**металл-окисел-полупроводник**
МП	**микропроцессор**
МПИ	**магистраль передачи информации**
МТ	**машина Тьюринга**
НГМД	**накопитель на гибких магнитных дисках**
НМД	**накопитель на магнитном диске**
НМЛ	**накопитель на магнитной ленте**
ОЗУ	**оперативное запоминающее устройство**
ОС	**операционная система**
ОУ	**операционное устройство**
ОШ	**общая шина**
П	**подложка (вывод транзистора)**
ПДП	**прямой доступ в память**
ПЗУ	**постоянное запоминающее устройство**
ПК	**персональный компьютер**
ПО	**программное обеспечение**
ПОЛИЗ	**польская инверсная запись**
ППЗУ	**программируемое постоянное запоминающее устройство**
ПС	**перевод строки (символ алфавитно-цифрового кода)**
ПУ	**передача управления**
РА	**расширенная арифметика**
РАФОС	**операционная система с разделением функций**
РВ	**реальное время**
РВР	**разделение времени**
РД	**регистр данных**
РК	**регистр команд**
РНП	**регистр начального пуска**
РОН	**регистр общего назначения**
РС	**регистр состояния**
РТЛ	**резисторно-транзисторная логика**
С	**сток (вывод транзистора)**
СА	**система адресации**
САПР	**система автоматизированного проектирования**
СМ	**система малых ЭВМ**