Е. А. Еремин

ПОПУЛЯРНЫЕ ЛЕКЦИИ ОБ УСТРОЙСТВЕ КОМПЬЮТЕРА

- Функциональные узлы ЭВМ и их взаимодействие
- Организация памяти и методы доступа к ней
- Обмен информацией с внешними устройствами
- Роль программного обеспечения

+ CD-ROM

ОСНОВЫ ИНФОРМАТИКИ

Е. А. Еремин

ПОПУЛЯРНЫЕ ЛЕКЦИИ ОБ УСТРОЙСТВЕ КОМПЬЮТЕРА

Санкт-Петербург
«БХВ-Петербург»
2003

УДК 681.3.06(075.3)
ББК 32.973я721
Е70

Еремин Е. А.

Е70 Популярные лекции об устройстве компьютера. — СПб.: БХВ-Петербург, 2003. — 272 с.: ил.

ISBN 5-94157-313-8

Книга представляет собой тщательно систематизированное, но в то же время популярное описание фундаментальных принципов устройства ЭВМ. Понятное и подробное изложение теории сопровождается большим количеством примеров, многие из которых оригинальны и достаточно необычны. После знакомства с очередной лекцией читателям предлагается провести на своем компьютере серию любопытных экспериментов. Они не просто иллюстрируют и дополняют лекционный материал, но и позволяют убедиться в том, насколько интересным и логически совершенным является наш ставший таким привычным электронный помощник.

Для широкого круга пользователей

УДК 681.3.06(075.3)
ББК 32.973я721

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Людмила Еремеевская*
Зав. редакцией	*Григорий Добин*
Редактор	*Ирина Радченко*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 11.07.03.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 21,9.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-313-8

Содержание

Предисловие

Данный курс лекций был первоначально задуман и разработан для педагогической газеты "Информатика", где имел название "Основы вычислительной техники" [29]. Главной идеей публикации было поддержать вытесняемую из школьного предмета "Основы информатики и вычислительной техники" вторую его составляющую, т. е. основы вычислительной техники. Для этой цели был тщательно подобран, проанализирован и систематически изложен материал по наиболее важным закономерностям устройства ЭВМ. Известно, что сведения по указанной теме обязательно есть практически в каждом учебнике информатики, но они невелики по объему и, чаще всего, поверхностны. Более того, нередки случаи, когда в погоне за упрощением или повышением наглядности изложения некоторые авторы используют описания, не вполне соответствующие фактическому устройству вычислительной техники.

Судя по отзывам, попытка отбора и обобщения материала оказалась удачной. Поэтому возникла идея сделать содержание лекций доступным более широкой аудитории читателей путем издания настоящей книги.

По сравнению с газетной версией, книга не просто поменяла название. В текст был внесен целый ряд изменений и дополнений, поскольку появилась возможность существенно увеличить объем излагаемого материала и рассмотреть большее число примеров. Наиболее существенным расширением стали появившиеся описания многочисленных экспериментов, которые иллюстрируют и дополняют теоретический материал. Проверяя изложенные в лекциях принципы, читатели смогут на собственном опыте убедиться, например, в справедливости упомянутого в каждом учебнике принципа двоичного представления и хранения информации, или узнать, сколько памяти содержит компьютер, на котором они работают, и достаточно ли ее для установленного программного обеспечения.

Отметим, что материалы лекций трактуются не только как форма проведения учебных занятий, а в более широком смысле. Известно, что в России

давно существовало такое явление, как публичные лекции, когда ученые или преподаватели высших учебных заведений рассказывали в популярной форме о достаточно сложных теориях и явлениях окружающего мира. Именно так автор и стремился построить свое изложение, всеми силами стараясь сделать материал интересным и понятным без каких-либо специальных предварительных знаний. Поэтому даже если читатель не является школьником или студентом, его не должно настораживать слово лекция в названии книги.

Пользуясь случаем, хочется выразить благодарность редакции газеты "Информатика" и лично ее редактору С. Л. Островскому за всестороннюю поддержку работы над первоначальным вариантом лекций.

Введение

Знать, как он работает, не менее важно, чем уметь работать с ПК. Вы можете вполне успешно пользоваться услугами компьютера, не понимая того, что в нем происходит. Однако чем глубже вы представляете процессы, происходящие в ПК, тем лучше будете использовать его возможности... Если что-нибудь случится в процессе работы с компьютером, вероятность того, что вы примите правильное решение, а не наделаете глупостей и не испортите все окончательно, будет выше.

П. Нортон

Важно... понимать, что эта вот привычка к пониманию логики вещей — сильная сторона российского образования. На эту привычку, умение, желание понять, разобраться в любом предмете в Америке сегодня есть большой спрос.
Я полагаю, что такой спрос и в Америке, и в России, и во всем мире с годами будет только увеличиваться.

А. Г. Кушниренко

Уважаемый читатель!

Вы держите в руках еще одну книгу о компьютерах. И, наверное, первый вопрос, который у вас возникает — чем она отличается от множества других?

Прежде всего, обратите внимание на тематику книги. Она посвящена внутреннему устройству компьютера, что немедленно дистанцирует ее от множества самоучителей, хороших и разных, часто обещающих вам немедленное освоение всех возможностей компьютера по приведенным в них рецептам и рекомендациям. Напротив, автор попытается помочь вам в постепенном понимании того, как работает компьютер, и это в свою очередь обеспечит плавный, но существенный рост ваших возможностей в грамотном его использовании.

С другой стороны, об устройстве компьютеров уже издано множество книг — справочников, пособий по выбору, ремонту и модернизации, технической литературы. Данная книга заметно отличается и от них. Во-первых, она написана в предположении, что читатель не имеет особой предварительной теоретической подготовки. Во-вторых, предметом обсуждения яв-

ляются не технические детали (успевающие порой устареть быстрее, чем появятся хорошие подробные книги о них), а фундаментальные, "проверенные временем" сведения, которые могут пригодиться при использовании компьютеров не только сегодня, но и, по крайней мере, в ближайшем будущем. В-третьих, данная книга не является сухим академическим изложением теории. Напротив, каждая лекция в дополнение к изложенному доступным языком материалу содержит большое количество практически полезных познавательных упражнений. Например: как определить объем памяти вашего компьютера и увидеть, достаточно ли ее; каким образом узнать расположение на диске только что сохраненного файла и влияет ли его фрагментация на скорость считывания данных, а также множество других подобных примеров. Перелистайте книгу — увидите сами.

Таким образом, остается лишь небольшая группа книг об устройстве компьютера, где доступным языком излагаются наиболее фундаментальные принципы его работы. Если дополнительно учесть уже упоминавшуюся демонстрацию практического применения этих знаний, то читатель, вероятно, согласится, что подобной литературы не так уж много. Определенную надежду на то, что эта книга получилась оригинальной, интересной и заслуживающей внимания широкой аудитории, дает автору его многолетний (начиная с 1974 года) опыт работы с вычислительной техникой, чуть менее короткий (с 1979 года) опыт преподавания многочисленных курсов по данной тематике и, наконец, достаточно большое количество написанных учебных пособий и публикаций образовательного характера (с 1985 по 2002 год их вышло в свет более сорока).

Следующая не менее важная характеристика книги — кому она может быть полезна? Любому человеку, интересующемуся компьютерами. Важно только, чтобы он действительно испытывал положительные эмоции, сидя за дисплеем, и у него хотя бы иногда возникали при этом вопросы "Почему?", "Зачем?" и "Как?" Особый интерес книга представляет для школьников и студентов, а также для преподавателей учебных заведений любого уровня. Иными словами, она вполне может служить обычной книгой для познавательного чтения, и в то же время, ее материал можно положить в основу учебного курса по соответствующему разделу информатики.

Еще раз подчеркнем, что предлагаемые вашему вниманию лекции призваны не столько разъяснять детальное устройство компьютера какой-либо определенной существующей архитектуры, сколько изложить наиболее общие базовые принципы и основную логику работы современных вычислительных машин. Иначе говоря, книга не претендует на освещение всевозможных технических деталей компьютера семейства IBM PC, процессора Pentium или программного обеспечения фирмы Microsoft для этого класса машин, хотя примеры в основном используются из этой области (в силу "национальных особенностей" компьютерного развития нашей страны триада IBM PC — Pentium — Windows является на данный момент существенно преобладающей).

Надеюсь, читатели уже составили некоторое представление о том, что они смогут узнать, познакомившись с этой книгой, равно как и о том, что в ней заведомо не стоит искать. А теперь несколько слов, как построено изложение материала в лекциях.

Первая лекция имеет вводный характер. В ней подробно рассматривается, насколько полезны знания устройства и логики работы компьютера для тех, кто его использует. Это единственное место в книге, где автор явным образом пытается убедить читателей в необходимости понимания основ функционирования вычислительной техники. В этой же лекции дается объяснение тому, по каким критериям отбирался материал для курса. Хотя все вышеизложенные вопросы в известном смысле являются вспомогательными, автор все же советует не пропускать вводной лекции и ознакомиться с ней, тем более что она содержит несколько оригинальных интересных примеров и иллюстраций.

Вторая лекция является ключевой. Именно здесь анализируется структура ЭВМ, ее основные блоки и взаимодействие между ними. Фактически этот материал является фундаментом для всех последующих лекций.

Лекции 3—6 улучшат ваши познания по вопросам функционирования каждого отдельного узла компьютера. Их материал тесно связан между собой, а также с *лекцией 2*, поэтому в тексте присутствует множество перекрестных ссылок. Раскрывается назначение каждого устройства, принципы и особенности организации его работы. Большое внимание при этом уделяется влиянию взаимодействия отдельных устройств на производительность всей машины в целом.

Седьмая лекция является переходом от изучения аппаратной части к программной. Здесь рассматривается значение программного обеспечения в современном компьютере, его состав, подробно проанализирована роль операционной системы в организации работы оборудования ЭВМ. Отдельно разбирается процесс загрузки программного обеспечения в компьютер.

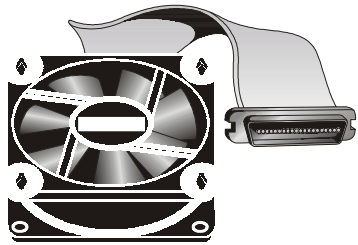
Наконец, *последняя лекция* посвящена учебным моделям компьютера. Ее основная идея состоит в том, что далеко не всегда удобно изучать принципы работы вычислительной техники на основе реальной ЭВМ. В лекции рассматривается целый ряд учебных моделей, включая самую современную "модель тысячелетия", предложенную классиком фундаментальной литературы по алгоритмам Д. Кнутом. Проводится сравнительный анализ достоинств и недостатков моделей.

Каждая из перечисленных выше лекций начинается с изложения теоретического материала, который составляет большую часть лекции. Затем формулируются основные выводы, и предлагается перечень вопросов, направленных на осмысление изложенного материала. Завершает большинство лекций описание любопытных экспериментов по теме лекции, которые читатели могут воспроизвести, следуя подробным рекомендациям книги. Хотя разбор

и повторение этих экспериментов не является обязательным, они, тем не менее, служат существенным дополнением для более глубокого усвоения содержания лекций.

Для проведения экспериментов всегда требуются соответствующие приборы и специализированные инструменты. Для данного курса их роль выполняет программное обеспечение. Во многих случаях удается воспользоваться стандартными средствами операционной системы, однако для организации более тонких и интересных экспериментов требуются более гибкие средства. Поэтому для некоторой части экспериментов приходится запускать специально разработанные для этой цели автором короткие программы на языке Паскаль (данный язык программирования обоснованно считается одним из самых наглядных и простых для понимания). В случае, если читатель не знаком с простейшими конструкциями Паскаля и не планирует в ближайшем будущем их изучать, для повторения описанных экспериментов можно запустить уже откомпилированный исполняемый файл с прилагаемого к книге CD-диска.

ЛЕКЦИЯ 1



Что мы будем изучать и почему

Наша первая лекция, как это обычно принято, имеет вводный характер. Прежде чем начинать изучение конкретных вопросов, полезно обсудить некоторые наиболее важные положения, относящиеся ко всему курсу в целом.

В первую очередь следует разобраться в том, какую роль играет данный курс в освоении компьютера. В каком-то смысле в лекции будет сделана еще одна попытка убедить сомневающихся читателей, а заодно и обосновать полезность знания фундаментальных основ работы вычислительной техники (ВТ). Возможно, некоторые читатели уже имеют четко оформившееся положительное мнение по этому вопросу — для них начальная часть лекции может показаться не совсем актуальной, и в этом случае ее можно прочитать не очень внимательно.

Следующий важный вопрос — это принципы отбора материала для данного курса. По мнению автора, ознакомившись с нижеизложенным, читатель сразу поймет, что он найдет в этой книге, а какой материал в нее сознательно не включен, и поэтому его лучше поискать в других изданиях.

Наконец, наиболее яркой и интересной обещает стать вторая половина лекции. В ней поставлена цель продемонстрировать, что, несмотря на огромную скорость развития вычислительной техники, ее фундаментальные основы во многом остаются неизменными. Помимо теоретического обоснования нашего курса, излагаемый материал содержит еще целый ряд оригинальных и интересных примеров, которые определенно привлекут внимание читателей. Обычно популярные книги стараются всячески поразить нас стремительными изменениями в мире компьютеров. В данной же лекции, напротив, автор призывает обратить внимание на другую сторону процесса и удивляться тому, что многие принципы до сих пор не изменились со времен изобретения самых первых ЭВМ.

Примечание

В тексте книги не делается особого различия между терминами "компьютер" и "ЭВМ", поскольку разница эта достаточно условна и, по крайней мере, при изложении наиболее универсальных закономерностей практически незаметна.

1.1. О возможных подходах к изучению компьютера

Давайте начнем разговор с того, что определим для себя, стоит ли вообще изучать устройство компьютера, и если да, то как это лучше делать. Обсуждение этого вопроса требуется непременно, поскольку даже многие из тех, кто использует компьютер более или менее регулярно, не считают знание основ его работы необходимостью. Вот как образно характеризует ситуацию один из известных авторов компьютерной литературы Чарльз Петцольд (Charles Petzold) [67]:

> "Приходилось мне слышать и такое: “Народу нет дела до того, как работают компьютеры”. И это, по-моему, тоже верно. Мне, например, доставляет удовольствие узнавать, как работает тот или иной прибор. Но при этом я предпочитаю выбирать, в чем я хочу разбираться, а в чем — нет. Мне, скажем, трудно без насилия над собой объяснить принцип действия холодильника.
>
> И все же люди часто задают вопросы, которые подтверждают их интерес к внутреннему устройству компьютера. Один из них: “В чем разница между оперативной и постоянной памятью?”
>
> Это, без сомнения, очень важный вопрос. На подобных понятиях основан весь рынок персональных компьютеров. Предполагается, что даже новичок понимает, сколько *мегов* в одной и *гигов* другой потребуется его программам. Считается также, что в самом начале знакомства с компьютером пользователь постигает концепцию файла и представляет себе, как файлы загружаются из постоянной в оперативную память, и затем из оперативной записываются обратно в постоянную".

Процесс освоения компьютера у каждого человека происходит по-своему, следовательно, единого универсального ответа на вопрос "нужно ли представлять себе устройство компьютера", вероятно, не существует. Мнения на этот счет сильно, порой диаметрально, расходятся.

Существует большое количество людей, которым принципиально все равно, как работает компьютер. В основу своего взаимодействия с ним они кладут один-два любимых самоучителя или справочника, советы грамотных друзей и, конечно же, хорошую память. Если сегодня мы зайдем в какой-либо из книжных магазинов, то увидим множество книг (и даже серий книг), в заголовках которых стоят слова "без проблем", "за 14 дней", "для занятых",

"одним взглядом" и множество аналогичных. Ничуть не пытаясь ругать или отвергать эти книги (в конце концов, раз их выпускают в таком количестве, значит кому-то они полезны!), давайте все же задумаемся, в чем их основная суть. Мне кажется, вы со мной согласитесь, что главный прием, с помощью которого такого рода книги учат нас, наиболее кратко и емко характеризуется словом "рецепт". В самом деле, каждый фрагмент, начинающийся со слов "для того, чтобы выполнить действие X, необходимо..." и содержащий далее детальный перечень конкретных шагов, есть не что иное, как рецепт (или, если хотите, более красиво, алгоритм) достижения поставленной цели. В случае, когда такой рецепт удается найти (в памяти пользователя, в книге, в конспекте лекций всевозможных курсов или в виде совета знающего друга), любой, даже самый малоопытный человек довольно успешно справляется с работой, если он просто будет внимательно следовать "руководящим указаниям".

В каких случаях рецептурный подход дает удовлетворительные результаты? Когда человек пользуется компьютером время от времени и разнообразие его действий невелико: несколько режимов работы в одной-двух программах. Когда программы, с которыми он работает, практически не меняются или меняются крайне редко. Наконец, когда у него постоянно есть рядом более опытный коллега, способный в любую минуту ответить на вопрос или помочь разрешить возникшую трудность. Довольно много профессий, начиная от бухгалтера и кончая писателем, могут удовлетворять перечисленным условиям.

Тем не менее, я глубоко убежден, что подобный подход сильно ограничен, поэтому то и дело такие пользователи что-то делают неправильно, причем порой необратимо неправильно, так что результаты работы фактически теряются. Особенно часто это случается в нестандартных или неожиданных ситуациях: изменение настроек используемой программы другими пользователями, нехватка места на диске, отсутствие файла, сбой компьютера и т. п. Кроме того, не всегда удается найти в книге "рецепт" именно для того действия, которое вы собираетесь сделать. Часто имеющееся описание отличается от ситуации на вашем компьютере какой-то незначительной, на первый взгляд, деталью, но именно она мешает успешно завершить операцию. Иногда не совсем четко сформулировано, как именно нужно выполнить тот или иной шаг. И это далеко не самый полный перечень.

Вот простой и вполне жизненный пример. Человека научили загружать картинки в графический редактор Paint, наносить на них некоторые поясняющие тексты и сохранять результаты изменений обратно на диск. Первая партия картинок уже обработана, и все идет замечательно. А вот со второй неожиданно возникают проблемы: пользователь делает все точно так же, но машина упорно "не желает" сохранять исправленный рисунок, ссылаясь на отсутствие доступа к файлу (между прочим, речь идет не о какой-нибудь древней версии операционной системы, а о Windows 2000!) Так в чем тут дело? Оказывается, вторая партия картинок была скопирована с компакт-

диска, и поэтому все они автоматически получили статус "только чтение", что, естественно, препятствует внесению изменений. В результате прекрасно работавший до этого рецепт оказывается бессильным. А что касается плохой диагностики ошибок в современном программном обеспечении, на которое хочется списать возникшую трудность, так это, к сожалению, скорее правило, чем исключение.

Кстати, именно на поверхностное понимание работы компьютера зачастую рассчитаны вирусы. Вспомните, например, один из нашумевших почтовых вирусов, который содержал фотографию известной российской теннисистки Анны Курниковой [И-6]. На деле такая "фотография", прилагаемая к письму, была исполняемым файлом "с двойным расширением" AnnaKournikova.jpg.vbs. Расчет был во многом сделан на то, что многие почтовые программы не отображают расширения приложенных файлов: в результате пользователи видели интригующую подпись AnnaKournikova.jpg, и многие из них в прямом смысле слова собственноручно запускали вирусную программу. А между тем увидеть истинный тип файла при желании совсем нетрудно — достаточно стандартным образом, используя правую кнопку мыши, посмотреть его свойства.

И еще один пример. Допустим, в качестве упражнения по освоению электронной таблицы Excel вы решили проверить тот факт, что $n! = (n - 1)! * n$. Примерное решение задачи может выглядеть так, как показано в табл. 1.1.

Таблица 1.1. *Пример работы с электронной таблицей Excel*

A1 = 5		C1 = ФАКТР(A1)	
A2 = A1 — 1	B2 = ФАКТР(A2)	C2 = B2 * A1	
		C3 = C1 — C2	D3 = ЕСЛИ(C1 — C2 = 0; "да"; "нет")

Примечание

На всякий случай напомним читателю, что запись *n*!, в математике называемая "красивым" термином *факториал*, есть просто произведение последовательных целых чисел от 1 до *n*. Например, 5! = 120.

Аккуратно внесем формулы в электронную таблицу (можно использовать файл overflow.xls, подготовленный на прилагаемом к книге CD-диске в каталоге под номером 1), и Excel послушно подтвердит, что все правильно, написав "да" в ячейке **D3**. А если взять другое число, допустим, 25? К нашему удивлению, появится результат, изображенный на рис. 1.1.

Неожиданным здесь является то, что ячейка **C3**, вычисленная как разность **C1 — C2**, равняется нулю, но проверка в клетке **D3** утверждает, что это не так! Эффект имеет простое объяснение, но только при условии понимания принципов хранения и обработки чисел в ЭВМ. Все дело в том, что 25! до-

вольно большое число и в ходе вычислений становится недостаточно имеющейся разрядной сетки для полного сохранения результатов; приходится последние (наименее важные) разряды округлять.

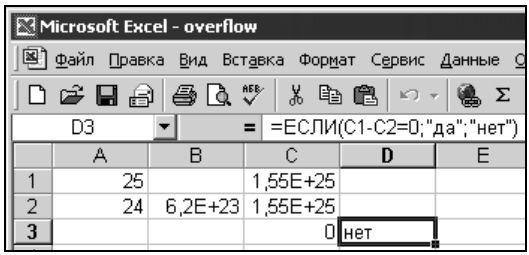


Рис. 1.1. "Странный" результат вычислений при $n = 25$

Итак, пусть мы поняли, что "выросли" из работы по готовым рецептам и хотим научиться более эффективной работе с компьютером. Какой же путь выбрать взамен? В качестве альтернативы можно предложить изучение наиболее общих закономерностей работы компьютера и его логики. Понимание тех принципов, на которых построено компьютерное оборудование и управляющее им программное обеспечение, позволит вам более грамотно анализировать возникающие в ходе работы трудности, а значит, и преодолевать их. Перечитайте еще раз цитату, вынесенную в качестве эпиграфа к книге — она принадлежит перу человека-легенды Питера Нортона, создателя известной каждому пользователю IBM PC программы Norton Commander, а также автору множества не менее известных во всем мире популярных книг о работе компьютера. Согласитесь, к его мнению стоит прислушаться.

В идеале понимание принципов работы компьютера должно приводить к полной ненужности "рецептов": используя свои знания и опыт, человек сам приобретает способность формулировать для себя такие "рецепты". Причем, особенно важно, что они не будут едиными на все случаи жизни, как в справочнике, а полностью соответствующими конкретной сложившейся ситуации. Вы полагаете, это невозможно? А как же тогда ваши знакомые советчики, эти своеобразные "ходячие энциклопедии"? Неужели вы думаете, что они все свои рекомендации помнят наизусть?

Конечно, и на этом пути есть свои трудности и неудобства. Дело в том, что современный компьютер имеет достаточно сложное устройство, которое, к тому же, постоянно совершенствуется. Иными словами, материала для изучения много, и он все время меняется. Ситуация еще более осложняется тем, что обычно большая часть литературы по устройству компьютеров написана для людей, собирающих новые компьютеры или осуществляющих модернизацию старых. Отсюда в них рассматривается огромное количество технических деталей, которые рядовому пользователю, разумеется, не требуются. Как же тогда быть? Неужели мы выбрали тупиковый для нас путь?

Попробуем все же найти выход! Просто надо очень тщательно проанализировать имеющийся материал об устройстве компьютера, отобрать наиболее важные и существенные сведения, а второстепенные подробности и тонкости не принимать во внимание. К тому же, как будет показано в *разд. 1.3*, такие фундаментальные принципы как раз меняются гораздо реже, чем технические детали, что с наших позиций тоже выгодно — не придется слишком часто переучиваться.

Таким образом, путь освоения компьютера, кажется, стал понятен. Остается решить, как отобрать материал в соответствии со сформулированными выше принципами.

1.2. О содержании лекций

Итак, мы выяснили, что сведения для изучения устройства вычислительной техники нуждаются в тщательном отборе. В данном разделе будут описаны главные принципы, по которым осуществлялся подбор материала для наших лекций.

Начнем с того, что курс лекций направлен на знакомство с наиболее важными, наиболее фундаментальными принципами работы вычислительных устройств, т. к. именно они позволяют создать у рядового пользователя компьютера некоторую стройную взаимосвязанную картину. Данную особенность очень хорошо выразил один из авторов нескольких широко известных учебников информатики А. Г. Кушниренко. Он писал:

> "Я спрашиваю себя: могу ли я сформировать целостное представление о компьютерах и компьютерных технологиях, не объяснив, что такое Гигабайт или чем текстовый файл отличается от документа Word? Выясняется, что не могу. Без этого картина получается какая-то однобокая. Значит, я буду пытаться объяснить в своем учебнике, что такое Гигабайт, что такое ASCII и какие бывают способы представления текстовых файлов.
>
> Могу ли я построить целостное представление о компьютерах, не объяснив квантовой механики или двух режимов работы процессора Intel, — да, я могу без этого всего обойтись, и потому в моем учебнике этого скорее всего не будет. Но я могу себе представить, что разные режимы работы процессора Intel могут оказаться существенным элементом другой картины мира, представленной в другом учебнике".

В приведенной цитате упомянут и еще один существенный принцип отбора материала — технические детали не представляют интереса для общеобразовательного курса за исключением, может быть, немногочисленных иллюстрирующих изложение примеров. Действительно, существует большое количество особенностей устройства различных моделей компьютеров, но они интересны в основном специалистам. Само по себе знание многочисленных конкретных фактов и числовых характеристик в области компьютерного

оборудования ничего плохого, разумеется, не представляет. Важно только не потерять "за деревьями леса", т. е. видеть в этих разрозненных сведениях некоторую систему. Об опасности подмены систематических знаний набором несвязанных фактов замечательно сказал американский фантаст Рэй Бредбери в одной из наиболее любимых мною книг "451 градус по Фаренгейту":

> "Устраивайте разные конкурсы, например: кто лучше помнит слова популярных песенок, кто может назвать все главные города штатов или кто знает, сколько собрали зерна в штате Айова в прошлом году. Набивайте людям головы цифрами, начиняйте их безобидными фактами, пока их не затошнит, — ничего, зато им будет казаться, что они очень образованные. У них даже будет впечатление, что они мыслят, что они движутся вперед, хоть на самом деле они стоят на месте".

В качестве дополнения к обоим сформулированным выше принципам приведем еще один важный тезис. Технические детали устройства ЭВМ устаревают необычайно быстро, а фундаментальные принципы, напротив, используются в течение длительного времени. Достаточно для примера сказать, что базовые идеи построения вычислительных устройств, выдвинутые известным математиком Джоном фон Нейманом с группой соавторов в 1946 году, по-прежнему сохраняют свою актуальность, несмотря на то, что сменилось уже три поколения ЭВМ!

Вопросам кажущегося противоречия между бурным развитием вычислительной техники и необыкновенной стабильности основных принципов ее устройства посвящается следующий раздел.

Наконец, еще один принцип, часто используемый автором в данной книге, состоит в описании тех или иных особенностей устройства ЭВМ в контексте их исторического развития. Подобный подход вполне оправдан: как замечательно писал уже цитировавшийся в начале лекции Ч. Петцольд, он очень полезен для понимания современного состояния дел:

> "Компьютеры наших дней сложнее тех, что появились 25 или 50 лет назад, но в основе своей они остались теми же. Вот почему изучать историю технологии так удобно: чем дальше вы уходите в прошлое, тем проще становится технология. Рано или поздно вы достигаете этапа, разобраться в котором уже не представляет особого труда".

Таковы наиболее важные принципы отбора материала для всех лекций нашего курса. Подчеркивая важность этих принципов, хочется в заключение привести еще одну известную и очень подходящую к случаю цитату из художественной литературы. В своем рассказе "Этюд в багровых тонах" классик логического детектива Артур Конан Дойл писал:

> "Мне представляется, что человеческий мозг похож на маленький пустой чердак, который вы можете обставить, как хотите. Дурак натащит туда всякой рухляди, какая попадется под руку, и полезные, нужные вещи уже

некуда будет всунуть, или в лучшем случае до них среди всякой завали и не докопаешься. А человек толковый тщательно отбирает то, что он поместит в свой мозговой чердак. Он возьмет лишь инструменты, которые понадобятся ему для работы, но зато их будет множество, и все он разложит в образцовом порядке. Напрасно люди думают, что у этой маленькой комнатки эластичные стены и их можно растягивать сколько угодно. Уверяю вас, придет время, когда, приобретая новое, вы будете забывать что-то из прежнего. Поэтому страшно важно, чтобы ненужные сведения не вытесняли собой нужных".

1.3. Стремительное развитие ВТ и стабильность ее фундаментальных принципов

1.3.1. Темпы развития ЭВМ. Поколения ЭВМ

Развитие вычислительной техники в литературе принято характеризовать понятием *поколение ЭВМ*. Каждое новое поколение представляет собой существенный шаг вперед и открывает перед пользователями новые горизонты и перспективы. Однако никогда не образуется избытка вычислительной мощности: возникают новые задачи и опять требуется улучшение характеристик машин.

В основе существующего деления на поколения лежит элементная база, на которой строятся ЭВМ (табл. 1.2).

Таблица 1.2. *Поколения ЭВМ*

Поколение	Элементная база	Способ общения
1	Электронные лампы	Двоичные коды
2	Транзисторы	+ языки программирования
3	Микросхемы	+ язык управления заданиями
4	БИС	+ пользовательский интерфейс

Тема поколений ЭВМ подробно обсуждается в каждом учебнике информатики, поэтому разрешите здесь не повторять в очередной раз одни и те же описания. За подробностями можно обратиться, например, к книге [78]. Отметим только, что довольно четкая смена поколений, происходившая раньше примерно раз в десять лет, сейчас замедлилась. Не очень ясна и картина с пятым поколением ЭВМ, созданием которого занимались японские ученые: несмотря на огромную проделанную ими работу, мы по-

прежнему продолжаем пользоваться микропроцессорными машинами четвертого поколения. По-видимому, эти факты лишний раз подтверждают некоторую искусственность всяческих классификаций, хотя, разумеется, сомнений в том, что славная летопись поколений будет со временем продолжена и дополнена, не возникает.

Обязательно обратите внимание на последний столбец табл. 1.2. Развитие вычислительной техники сопровождается отчетливым совершенствованием способов общения человека с машиной и созданием новых все более удобных устройств ввода и вывода данных. Происходит постепенная замена языка машины (двоичные коды) на все более близкие человеку формы обмена информацией.

Чтобы показать огромную скорость развития в производстве вычислительной техники, обычно приводится масса разнообразных сравнений и фактов. Например, в 1993 году мощность первых процессоров Pentium превышала вычислительные возможности NASA на момент высадки космонавтов на Луну. Автору, чтобы не повторяться, хочется привести какой-нибудь оригинальный пример. Перед вами на рис. 1.2 фотография ячеек ЭВМ трех разных поколений (см. слева направо): на лампах, на транзисторах и на микросхемах; надеюсь, качество фотографии позволяет вам хорошо рассмотреть последнюю маленькую ячейку в виде "многоногого жучка". Подчеркнем, что с точки зрения выполняемых ими функций все ячейки абсолютно одинаковы — каждая из них представляет собой два триггера (*триггер* — это электронная схема для хранения одной минимальной единицы информации, т. е. 1 бита).

Фотография фактически является наглядной диаграммой уменьшения размеров элементов вычислительной техники для первых трех поколений (авторучка специально находится в кадре для того, чтобы дать читателям возможность иметь некоторый характерный масштаб изображения; для этой же цели в качестве фона подложена миллиметровая бумага). Понятно, что вычленить два триггера из современного микропроцессора четвертого поколения не представляется возможным: для этого надо из кристаллика длиной порядка 1 см выделить ничтожную его часть. Даже если бы это каким-то чудесным образом удалось, камера, с помощью которой сделана данная фотография, все равно не сумела бы запечатлеть столь крошечный объект.

Как уже было сказано, показанные на фотографии ячейки содержат триггеры. Поскольку триггер является важным цифровым устройством, способным хранить один бит информации, мы еще неоднократно встретимся с ним в нашем курсе. А пока в ознакомительных целях приведем на рис. 1.3 схему некогда распространенной микросхемы К1ТК552 (два D-триггера) [74], фотография которой, кстати, и была приведена на рис. 1.2. Левая часть схемы реализует определенную входную логику, а правая, состоящая из двух симметричных узлов, образует собственно триггер. На выходе **Q** отражается хранящаяся в триггере информация: 0 или 1.

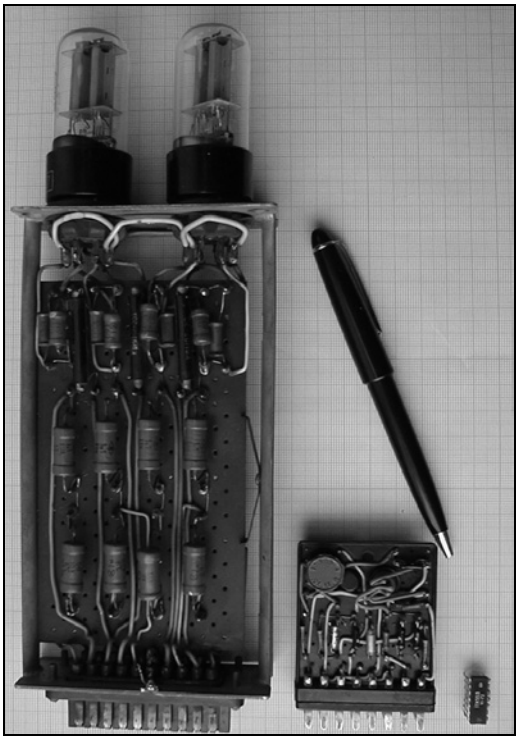


Рис. 1.2. Уменьшение размеров ячеек ЭВМ первых трех поколений

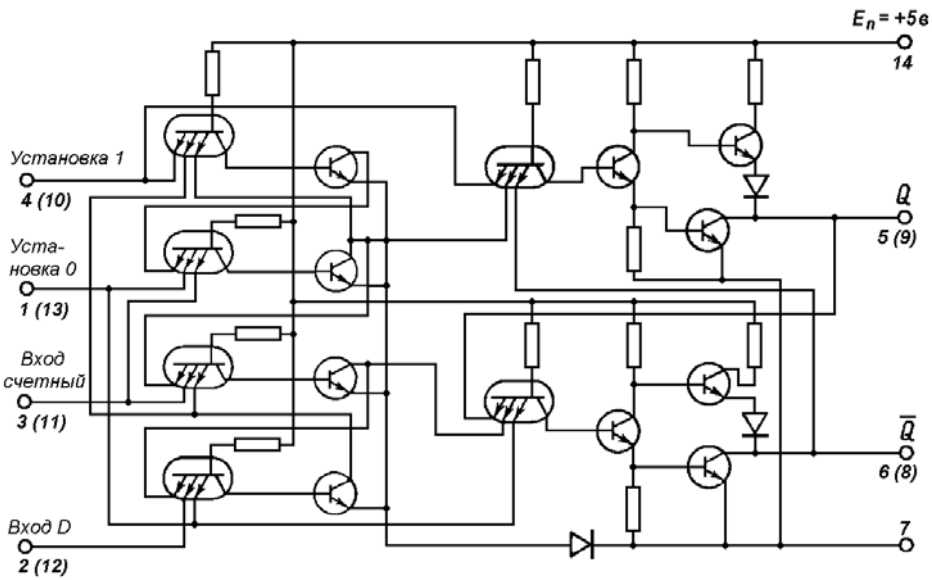


Рис. 1.3. Принципиальная схема интегрального триггера

Итак, темпы развития технологии производства компонентов вычислительной техники поистине впечатляющи. А насколько изменяются фундаментальные принципы построения ЭВМ? Сравним между собой две достаточно типичные вычислительные машины, с которыми автору приходилось лично иметь дело. Одна из них, ЭВМ модели Наири-2, принадлежала ко второму поколению и была в свое время достаточно распространена. Первый сеанс общения с этой машиной у автора состоялся в 1974 году; уже более десяти лет прошло с тех пор, как ее разобрали на составные части (кстати, из главного блока после небольшой переделки вышел замечательный стол огромных размеров). В качестве второго компьютера, с которым мы будем сравнивать, возьмем обычный компьютер семейства IBM PC с процессором Intel 80486. Такие машины еще вполне можно увидеть, хотя, разумеется, это далеко не последняя модель. Вы спросите почему не Pentium? Да потому, что система команд процессоров все больше расширяется, но в основном за счет достаточно специфических операций. Все это только смазывает картину, хотя и принципиально не меняет ее (см. прим. 1 к табл. 1.4).

1.3.2. Система команд типичной ЭВМ второго поколения

ЭВМ Наири-2 [55] была разработана в 1968 году и серийно производилась в Ереване. Ее внешний вид показан на рис. 1.4: машина состояла из двух крупных по размеру блоков, причем слева на стойке с блоками питания стоят периферийные устройства (электрифицированная пишущая машинка, фотосчитыватель с бумажной ленты и перфоратор для ее вывода), а собственно машина находится на фотографии справа. Обязательно обратите внимание на длинные ряды лампочек в центральной верхней части машины; для удобства считывания они были разбиты на группы с помощью колпачков белого и черного цвета и отображали двоичное состояние внутренних регистров процессора ЭВМ.

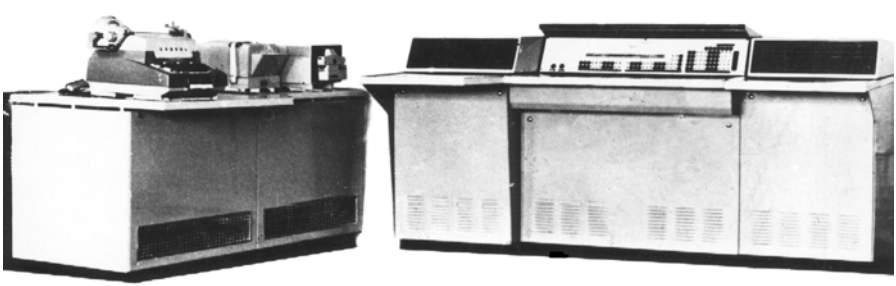


Рис. 1.4. Вид ЭВМ второго поколения Наири-2

Наири-2 могла работать с целыми и вещественными числами (стандартные вещественные числа хранились в одной ячейке, а длинные — в двух после-

довательных ячейках). В систему ее команд входило 102 инструкции, из них 47 — собственно машинные операции (табл. 1.3), а остальные — так называемые псевдооперации, реализуемые при помощи специальных подпрограмм в долговременном запоминающем устройстве — ДЗУ. (Интересно вспомнить, что ДЗУ было выполнено на ферритовых кольцах, причем информация в нем в прямом смысле слова "прошивалась" путем продергивания тонкого медного провода через сердечник при единице и мимо него при нуле.) Важное положение среди псевдоопераций занимали разнообразные действия над вещественными числами, которые мы для простоты изложения рассматривать не будем.

Таблица 1.3. *Полная система машинных операций ЭВМ Наири-2 (1968 г.)*

п, п1	Передача числа
с, с1, с2, с3	Сложение
в, в1, в2, в3	Вычитание
у, у1, у2, у3	Умножение
д, д1, д2, д3	Деление
в4	Вычитание модулей
l, l1	Логическое сложение (AND)
л, л1	Логическое умножение (OR)
м, м1	Сложение по модулю 2 (XOR)
а, а1	Сдвиг вправо (арифметический)
б, б1	Сдвиг влево (логический)
г, г1	Длинный сдвиг
н, н1	Нормализация
и, и1, и2, и3	Абсолютный переход с возвратом
е, е1, е2, е3	Относительный переход с возвратом
к, к1	Останов
ч	Чтение
о, о1	Обращение (вывод)
х	Холостая команда

Примечание

Большинство инструкций имеют несколько модификаций, которые обозначены цифровым индексом. Например, операция п копирует содержимое одной ячейки памяти в другую, а п1 сохраняет результат предыдущей операции в память.

Приведем также примеры некоторых (далеко не всех!) псевдоопераций ЭВМ Наири-2, которые находились в ее ДЗУ, состоявшем из 16 384 ячеек (команд). Это `ln` — вычисление логарифма, `sn` — вычисление синуса, `пп` — печать чисел с плавающей запятой, `пс` — печать содержимого (в двоичном виде), `пк` — печать команд, `сд`, `вд`, `уд`, `дд` — арифметические операции с длинными числами, `ск`, `вк`, `ук`, `дк` — действия над комплексными числами, `Га` — вычисление гамма функции, `mn` и `mx` — нахождение минимума и максимума в массиве и многие другие.

1.3.3. Система команд типичной ЭВМ четвертого поколения

А теперь для сравнения приведем систему команд хорошо знакомого всем процессора Intel 80486, который и поныне все еще продолжает служить людям в некоторых не слишком новых компьютерах. На рис. 1.5 можно увидеть фотографию главной платы компьютера на базе процессора Intel 80486.

Табл. 1.4 построена по данным монографии [7].

Рис. 1.5. Вид главной платы компьютера на базе процессора Intel 80486

Таблица 1.4. *Основные команды микропроцессора Intel 80486*

MOV	Пересылка данных
PUSH, PUSHA, POP, POPA	Чтение/запись в стек
LEA, LDS, LES, LFS, LGS, LSS	Загрузка адреса и селекторов
MOVSX, MOVZX	Пересылка с расширением знака
XCHG	Обмен содержимым
XLAT	Преобразование кодов
BSWAP	Перестановка байтов
ADD, ADC, XADD, INC, AAA, DAA	Сложение
SUB, SBB, DEC, CMP, CMPXCHG, NEG, AAS, DAS	Вычитание
MUL, IMUL, AAM	Умножение
DIV, IDIV, AAD	Деление
CBW, CWDE, CWD, CWQ	Преобразование разрядности данных
NOT	Логическое НЕ
AND	И
OR	ИЛИ
XOR	Исключающее ИЛИ
TEST	Логическое сравнение операндов
SHL, SAL	Логический и арифметический сдвиг влево
SHR	Сдвиг вправо
SHLD, SHRD	Двухоперандные сдвиги
ROL, ROR, RCL, RCR	Циклические сдвиги
BT, BTS, BTR, BTC, BSF, BSR	Операции с битами
LODS, STOS, INS, OUTS, MOVS, CMPS, SCAS	Операции со строками
JMP	Безусловный переход
JA/JNBE, JAE/JNB/JNC, JB/JNAE/JC, JBE/JNA, JE/JZ и др.	Условные относительные переходы
CALL, RET, INT, IRET	Переход к подпрограммам и возврат
SETA/SETNBE, SETAE/SETNB и др.	Условная установка байта
LOOP, LOOPE/LOOPZ, LOOPNE/LOOPNZ, JCXZ	Организация циклов

Таблица 1.4 *(окончание)*

MOV	Пересылка данных
IN	Ввод из порта в аккумулятор
OUT	Вывод из аккумулятора в порт
HLT	Останов
NOP	Отсутствие операции

Примечания

- Для краткости в таблицу не включены некоторые группы команд, связанные с управлением процессором, организацией защиты памяти, поддержкой языков высокого уровня и некоторые другие. Кстати, именно эти инструкции являются максимально машинно зависимыми, т. е. существенно зависят от модели процессора. Иными словами, с точки зрения общеобразовательных целей они представляют гораздо меньший интерес.
- Особо следует сказать об операциях с вещественными числами. В процессорах фирмы Intel для выполнения этих действий используется специальный блок — так называемый математический сопроцессор. Начиная с модели 486, он изготовляется в том же самом кристалле, что и основной процессор. Команды сопроцессора в таблице также не отражены.

Перейдем к анализу табл. 1.3 и 1.4.

1.3.4. Сравнение систем команд ЭВМ второго и четвертого поколений

А теперь самая главная и интересная часть — сравним между собой табл. 1.3 и 1.4. Напомним читателю, что они разделены огромным по меркам развития вычислительной техники временем — более 20 лет. Серым цветом в таблицах выделены клетки, содержащие аналогичные операции. И как ни удивительно, они охватывают большую часть анализируемых таблиц! Кроме того, не следует забывать, что процессоры фирмы Intel относятся к категории CISC-процессоров (Complex Instruction Set Computer), т. е. процессоров с расширенной системой операций. А это значит, что для многих других процессоров результаты аналогичного сравнения будут еще более близкими.

Примечание

Читателям, которые усомнятся в вышеизложенном по причине того, что 486-й процессор уже устарел, советую обратиться к любой книге по процессорам Pentium [14] и убедиться, что появившиеся в последних процессорах новые команды принципиально не изменяют ситуации.

Итак, получаем важный и интересный вывод: ядро системы команд, образуемое такими группами операций, как перепись информации, арифметические и логические команды, сдвиги и переходы, за несколько поколений (!) ЭВМ практически не изменилось. Неужели вас, уважаемые читатели, это не удивляет?!

Еще одно наблюдение состоит в том, что дополнительные команды процессоров Intel (см. табл. 1.4) в основном расширяют возможности программиста, а не предоставляют принципиально новых возможностей. Скажем, наличие группы команд для организации циклов во многих случаях облегчает написание программы, но, с другой стороны, любой цикл можно реализовать и без них, с помощью инструкций условных переходов. Или операции над отдельными битами, которые всегда можно выполнить на базе стандартных логических операций.

Пожалуй, наиболее важным прогрессом в развитии ЭВМ является переход к обработке значительно большего количества различных данных, который как раз не очень заметен из сравнения обсуждаемых таблиц. Хотя ЭВМ первых поколений и могли работать с несколькими видами информации (в основном числами) и даже имели "зачаточные" возможности по выводу отдельных символов, развитые средства обработки строковых данных там отсутствовали. Работа со звуком и видео в те времена, естественно, даже не обсуждалась. Основу памяти таких машин составляла ячейка под стандартное число или команду (в Наири ее длина равнялась 36 двоичным разрядам), поэтому данные другой длины не могли при такой организации рационально храниться.

Обязательно обратите внимание на появление в табл. 1.4 новых по сравнению с табл. 1.3 групп команд преобразования данных: пересылка с расширением знака, перестановка байтов, преобразование разрядности данных, операции со строками. Причина этих нововведений состоит в том, что память современных компьютеров имеет значительно более гибкую байтовую структуру, о которой мы обязательно подробно поговорим в следующих лекциях.

И последнее замечание, которое касается обеих анализируемых таблиц. Обратите внимание на простоту команд, которые непосредственно могут исполняться процессором даже современной ЭВМ. Данный факт интересен тем, что сложное программное обеспечение, содержащее тысячи таких простых команд, как сложение, сравнение и др., может вести себя настолько разумно, что как-то забывается, что в его основе лежат все те же несложные базовые инструкции процессора.

1.4. Некоторые выводы

Итак, подведем итоги того, что мы узнали из первой лекции.

- Понимание основных принципов работы такого достаточно сложного в обращении и настройке устройства, как компьютер, полезно любому человеку, систематически его использующему.
- Подход, условно названный в лекции рецептурным, когда все манипуляции с компьютером выполняются в соответствии с заранее подготовленными кем-то описаниями, хорошо работает лишь в простейших случаях. Для выполнения квалифицированной работы на компьютере его ограниченность отчетливо видна.
- Необходимость в практическом применении знаний о фундаментальных основах ВТ возникает не всегда. Чем более необычна для пользователя ситуация, тем острее он нуждается в подобных знаниях, и особенно в умении их анализировать.
- В основе функционирования современного компьютера лежит большое количество идей, теорий, принципов и технических решений. Для того чтобы лучше ориентироваться в них, необходимо сформировать для себя некоторое систематическое представление о работе компьютера. Следовательно, очень важно суметь выделить те наиболее существенные принципы и факты, которые необходимо хорошо знать и достаточно глубоко понимать.
- Изучение тех или иных принципов устройства ЭВМ в их историческом развитии часто оказывается очень полезным для понимания современного состояния дел. К тому же, чем дальше вы уходите в прошлое, тем проще становится разобраться в рассматриваемых технологиях и понять суть дела.
- Несмотря на огромный технический прогресс, который увеличил на несколько порядков быстродействие ЭВМ и соответственно уменьшил их размеры, фундаментальные принципы построения ВТ в основном сохранились.
- Как это не парадоксально звучит, в основе всех самых "умных" и изощренных программ лежит достаточно ограниченная система простых инструкций, которые непосредственно понимает процессор компьютера. Секрет состоит в универсальности этих команд и в возможности их комбинировать.
- На самом низком (аппаратном) уровне инженеры стараются создать компьютер максимально просто. Для адаптации машины к задачам, которые требуется решать на практике, используется сложное программное обеспечение. Очевидно, что установить новое программное обеспечение значительно проще, чем изменять конструкцию компьютера.

1.5. Вопросы для осмысления

В разделе с таким названием в данной и в последующих лекциях вам будет предложено подумать над проблемами, решения которых связаны с излагаемым материалом. Причем не всегда сформулированные вопросы имеют очевидные и однозначные ответы. Главная цель этого раздела — предложить читателям решить их тем или иным способом для себя. Автор полагает, что наличие собственной позиции по подобным вопросам и есть часть той самой систематической картины, к формированию которой мы договорились стремиться, разбираясь в сложном и запутанном внутреннем мире современного компьютера.

1. Попробуйте четко ответить на вопрос: насколько, по-вашему, полезно знание фундаментальных основ работы вычислительной техники?
2. Постарайтесь проанализировать свой опыт общения с компьютером и припомнить ситуации, когда понимание логики работы компьютера вам помогло принять правильное решение. А может быть, вам вспоминаются и обратные примеры?
3. Какие наиболее убедительные аргументы вы бы привели своему собеседнику-пессимисту в ответ на его традиционное "Кому все это нужно?" (автор убежден, что читатели найдут множество эффектных аргументов, которые не отражены в тексте лекции).
4. Сопоставьте эксплуатацию современного компьютера с другими устройствами, скажем, телевизором или автомобилем. Согласны ли вы с утверждением, что использование компьютера намного сложнее, чаще требует принятия решений и, следовательно, больше нуждается в понимании внутреннего устройства?
5. Проанализируйте реальную ситуацию, описанную в книге Л. Гринзоу [13]:

   "В офисе клиента, которого я консультировал, я однажды застал их директора по маркетингу за следующим занятием: он запускал Windows, программу-редактор, что-то выискивал, беспорядочно тыкая в разные кнопки на экране, после чего выключал компьютер из сети. Пока я наблюдал за ним, он повторил эту процедуру несколько раз. В конце концов я понял, что происходило: этот человек просто не знал, как удалить текст и начать новый документ. А включение/выключение компьютера просто оказалось ближайшим решением его проблемы. (Да, забыл упомянуть: клиентом являлась компьютерная компания.)" Как вы думаете, каких минимальных знаний о работе с компьютером не хватало описанному пользователю?
6. Еще одна цитата (Майкл Крайтон, роман "Стрела времени"): "Спросите себя... — каков преобладающий способ восприятия в конце двадцатого столетия? Какими люди видят вещи и какими они ожидают их увидеть?

Ответ прост. В любой сфере деятельности, от бизнеса до политики, от маркетинга до образования, преобладает восприятие через развлечение.

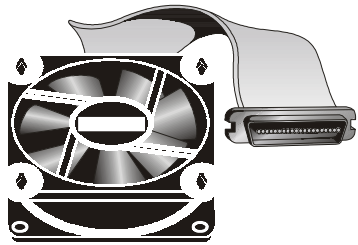
...Сегодня каждый ожидает, что его будут развлекать, причем будут развлекать постоянно. Деловые встречи должны происходить молниеносно, иметь очень краткую программу и сопровождаться анимационной графикой, чтобы участники не скучали. Торговые центры и магазины должны быть привлекательными, чтобы развлекать нас в той же мере, в какой и предлагать товары. Политические деятели должны обладать телегеничными лицами и сообщать нам только то, что мы хотим услышать. Школы должны проявлять крайнюю осторожность, чтобы не скучали молодые умы, ожидающие от обучения динамичности и разнообразия, присущих телевизионным программам. Студентов нужно удивлять — всех нужно удивлять, или же они отвернутся; отвернутся от марки товара, от канала телевидения, от партии, откажутся от лояльности.

Такова реальность интеллектуального бытия западного общества в конце столетия".

Насколько уместно развлечение в образовании? Существует ли грань, которую нельзя переступать, не рискуя превратить обучение в его видимость? Готовит ли такое "развлекающее" обучение к реальной жизни? Оцените с точки зрения рассматриваемого вопроса построение данной лекции.

7. Внимательно проанализируйте табл. 1.3 и 1.4. Попробуйте самостоятельно выделить основные группы операций, которые умеет выполнять ЭВМ.
8. В табл. 1.3 среди логических операций ЭВМ Наири-2 нет инструкции НЕ, зато имеется операция исключающее ИЛИ. Сможете ли вы догадаться, как с помощью последней инструкции можно заменить первую?

ЛЕКЦИЯ 2



Функциональные блоки ЭВМ

А теперь мы поговорим об устройстве ЭВМ, а точнее — об ее основных узлах, их назначении и принципах взаимодействия. В каком-то смысле эта лекция обрисует контуры данного курса, поскольку большая его часть посвящена обсуждению работы тех или иных блоков современного компьютера.

Во избежание путаницы сразу хочется обратить внимание читателей на следующее обстоятельство. Сейчас на прилавках магазинов представлено огромное количество книг по компьютерной тематике. Многие из них трактуют блоки компьютера с ***конструктивной*** точки зрения (системный блок, клавиатура, монитор и т. д.), т. е. описывают наиболее распространенные у нас модели IBM PC с точки зрения покупателя, которому важно знать, сколько коробок он должен получить в магазине. "Определение ликбезовских курсов, согласно которому компьютер состоит из системного блока, монитора, клавиатуры и манипулятора "мышь", естественно, никуда не годится — под него не подходят даже портативные компьютеры или монтируемые в стойку серверы с IBM PC-совместимой архитектурой". [45] С нашей точки зрения месторасположение узла в том или ином "ящике" является не очень существенным хотя бы потому, что разные модели машин действительно могут собираться на заводах по-разному. В дополнение к приведенной цитате можно вспомнить не так давно распространенные бытовые и школьные компьютеры (БК, Спектрум, Корвет, УКНЦ и т. п.), у которых клавиатура монтировалась на крышке системного блока, а не являлась самостоятельным устройством (рис. 2.1). Или компьютеры семейства Macintosh, у которых монитор и системный блок часто представляют собой единое целое (рис. 2.2). Нет, нас будут интересовать именно ***функциональные*** блоки — процессор, память всевозможных видов, накопители на дисках различной природы и т. д. вне зависимости от того, где и на каких платах они смонтированы сегодня или будут устанавливаться завтра.

Рис. 2.1. Вид бытового компьютера БК-0010: системный блок или клавиатура?

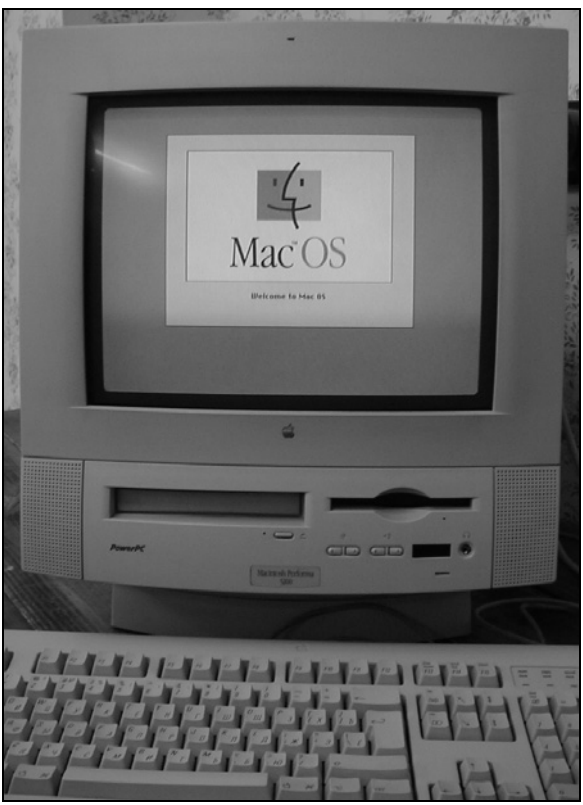


Рис. 2.2. Компьютер Macintosh, выполненный в виде моноблока (системный блок совмещен с монитором)

Несмотря на кажущуюся простоту, вопрос о функциональных составных частях ЭВМ, особенно об их взаимодействии, в различных учебниках может трактоваться несколько по-разному и, к сожалению, не всегда правильно с точки зрения соответствия действительности. Порой авторы в погоне за методически изящным изложением или аналогиями описывают функционирование ЭВМ достаточно "противоестественным" образом. Ярким примером такого рода является кочующее из книги в книгу положение о том, что информация всегда попадает с устройства ввода в ОЗУ напрямую, минуя процессор. На самом деле, такой режим возможен *(см. разд. 2.2.4)*, но он не только не является единственным, но и был разработан значительно позднее, чем стандартный способ чтения информации через внутренние регистры процессора.

2.1. Основные блоки ЭВМ

Как известно, впервые над устройством автоматической машины, способной работать без вмешательства человека по заранее составленной программе, задумался гениальный английский ученый Чарльз Бэббедж (сошлемся для примера на книгу [17]). Более полутора столетий назад он предложил и разработал для этой цели свою аналитическую машину, которая хотя и не была построена, но идеи по ее устройству получили впоследствии широкую известность. По мысли изобретателя, аналитическая машина должна была состоять из следующих частей [17]:

- устройство, в котором производятся все операции по обработке всех видов информации; по современной терминологии оно называется *арифметико-логическим устройством* (АЛУ);
- устройство, обеспечивающее организацию выполнения программы обработки информации и согласованное взаимодействие всех узлов машины в ходе этого процесса — *устройство управления* (УУ); АЛУ и УУ в настоящее время удается выполнить в виде единой интегральной схемы, которая называется *микропроцессором*;
- устройство, предназначенное для хранения исходных данных, промежуточных величин и результатов обработки информации, а также, что очень важно, самой программы обработки информации; данное устройство принято называть *запоминающим* (ЗУ) или часто для краткости просто *памятью* (по-видимому, все-таки точнее считать памятью совокупность всех ЗУ машины [33]); существуют различные виды памяти, в том числе оперативное запоминающее устройство (ОЗУ) и внешняя память на магнитных или оптических дисках;
- разнообразные устройства, способные преобразовывать информацию в форму, доступную компьютеру (часто этот процесс называют кодированием), — *устройства ввода*;

- и, наконец, устройства, преобразующие результаты обработки в доступную человеку форму — *устройства вывода.*

Мы видим, что несмотря на активные поиски в области создания все новых и новых вычислительных машин, большинство компьютеров по-прежнему построено согласно классическим идеям, предложенным по меркам истории ВТ очень давно.

Но что же тогда все-таки изменилось более чем за полвека? Прежде всего, бросается в глаза совершенствование технологий производства и, как следствие, резкое уменьшение размеров (помните рис. 1.2?) и повышение быстродействия вычислительной техники. Но не менее важно и еще одно изменение, которое мы сейчас рассмотрим подробнее. Это совершенствование структуры ЭВМ и изменение принципов взаимодействия ее основных узлов.

2.2. Взаимодействие блоков ЭВМ

2.2.1. Классическая структура ЭВМ

Как уже говорилось выше, классические принципы построения ЭВМ были систематически изложены группой авторов, среди которых был Джон фон Нейман. Огромный авторитет Неймана, который в то время был очень известным математиком и физиком, привел к тому, что ***всем*** базовым принципам построения вычислительной техники стали приписывать его авторство, а архитектура с последовательным выполнением команд получила название "фон-неймановской", хотя, по-видимому, личная заслуга Неймана здесь заметно преувеличена [35, 78].

Кратко сформулируем классические принципы устройства ЭВМ.

Использование двоичной системы для представления чисел. Нелишне напомнить, что все самые первые вычислительные машины хранили обрабатываемые числа в десятичном виде. В рассматриваемой статье были убедительно продемонстрированы преимущества двоичной системы для технической реализации (согласитесь, что намного проще создать устройство с двумя устойчивыми состояниями, чем с десятью), удобство и простота выполнения в ней арифметических и логических операций. В дальнейшем ЭВМ стали обрабатывать и нечисловые виды информации: текстовую, графическую, звуковую и др. Но по-прежнему двоичное кодирование данных составляет информационную основу любого современного компьютера.

Принцип "хранимой программы". Первоначально программа задавалась путем установки перемычек на специальной коммутационной панели. Это было весьма трудоемким занятием: например, для изменения программы одной из первых ЭВМ "ЭНИАК" требовалось несколько дней (в то время как собственно расчет мог продолжаться всего несколько минут, после чего выходила из строя одна из многочисленных ламп). Согласно рассматриваемому

принципу программа также должна храниться в виде набора нулей и единиц, причем в той же самой памяти, что и обрабатываемые ей числа. С точки зрения хранения и способов обработки принципиальная разница между программой и данными отсутствует.

Команды и данные помещаются в ячейки памяти, доступ к которым осуществляется по адресу. Адресом ячейки фактически является ее номер; таким образом, местонахождение информации в ОЗУ также кодируется в виде чисел (попутно подчеркнем, что адрес является целым и неотрицательным). В этом состоит так называемый ***принцип адресности***.

В построенной по фон-неймановским принципам ЭВМ происходит последовательное считывание команд из памяти и их выполнение. Номер (адрес) очередной ячейки памяти, из которой будет извлечена следующая команда программы, формируется и хранится в специальном устройстве — счетчике команд. ***Наличие программного счетчика*** также является одним из характерных признаков рассматриваемой архитектуры.

Фон Нейман с соавторами не только выдвинули основополагающие принципы логического устройства ЭВМ, но и предложили ее структуру, которая полностью воспроизводилась в течение первых двух поколений ЭВМ. Схема устройства такой ЭВМ представлена на рис. 2.3.

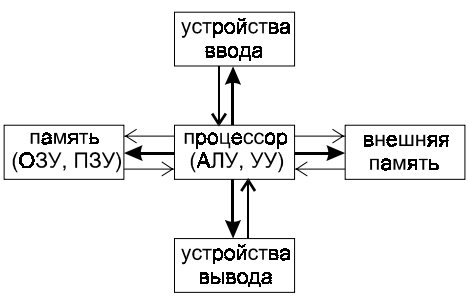


Рис. 2.3. Структура ЭВМ первого и второго поколения

Тонкими стрелками показаны направления движения информации, а толстыми зачерненными — управляющие воздействия УУ центрального процессора на все остальные устройства машины.

Все изображенные блоки нам уже знакомы. Следует только отметить, что внешняя память отличается от устройств ввода и вывода тем, что данные в нее заносятся в виде, удобном компьютеру, но недоступном для непосредственного восприятия человеком. Так, накопитель на магнитных дисках и CD-ROM устройства относится к внешней памяти, клавиатура и сканер — устройство ввода, а дисплей и печать — устройства вывода.

Наиболее существенной особенностью описанной выше реализации ЭВМ является центральное положение процессора в схеме. Во-первых, через него

проходят все информационные потоки, а во-вторых, он управляет согласованной работой всех устройств машины.

Автор считает тезис о центральном положении процессора в классической функциональной схеме принципиальным. Он особенно важен потому, что в некоторых книгах помещают в центр схемы ОЗУ. Но при этом получается, что информация вводится сразу в ОЗУ, а только затем попадает в процессор, что в обсуждаемых ЭВМ первого и второго поколения было просто невозможным.

2.2.2. Как это на самом деле работало

Данный раздел является дополнительным и при ознакомительном чтении может быть пропущен без особого ущерба. Он является своего рода доказательством тезиса о центральной позиции процессора, высказанного в предыдущем разделе. Для выяснения деталей взаимодействия процессора с внешними устройствами обратимся к анализу механизма ввода данных с перфоленты (рис. 2.4). На ЭВМ первых поколений такое устройство было широко распространено.

Рис. 2.4. Так выглядела перфолента — один из первых носителей информации

На рис. 2.5 схематически изображены ОЗУ, некоторые регистры процессора и в качестве вводимой информации — данные на перфоленте. Посмотрим, как это все взаимодействовало.

Примечание

Регистры процессора будут подробно рассматриваться в *разд. 3.1*. Пока достаточно понимать, что *регистр* — это устройство, находящееся внутри процессора и предназначенное для хранения и обработки информации.

В первую очередь опишем то, что нас в данный момент больше всего интересует: два регистра, через которые процессор взаимодействует с ОЗУ. Усло-

вимся их называть регистр адреса ОЗУ и регистр данных ОЗУ, что сразу поясняет соответствующее функциональное назначение: в первый заносится адрес требуемой ячейки памяти, а во второй считывается ее содержимое (или наоборот, двоичный код, помещенный в регистр данных, записывается в память).

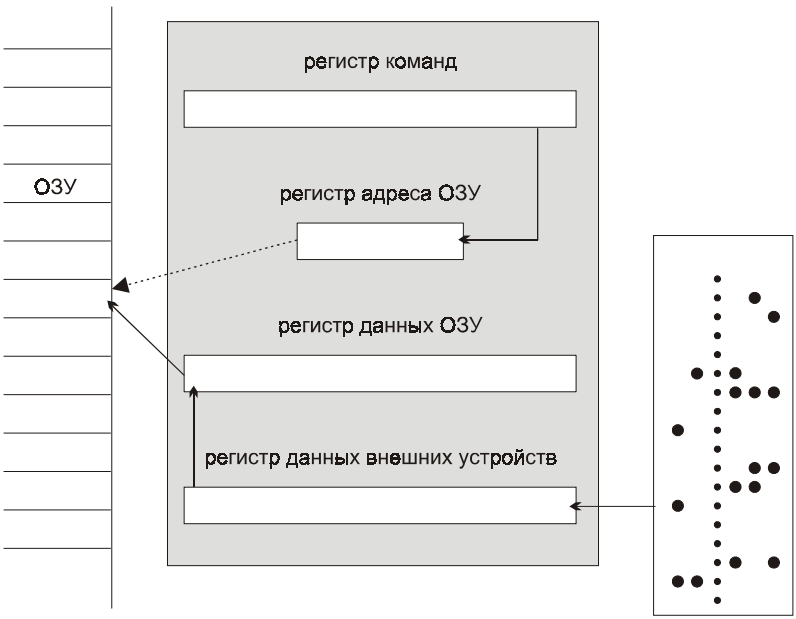


Рис. 2.5. Механизм загрузки данных с внешнего устройства в ОЗУ

Внимательные читатели, конечно, заметили, что регистр адреса нарисован в виде меньшего по размеру прямоугольника. Это не дефект рисунка, а, напротив, сознательное подчеркивание действительного факта. Так сложилось, что для машин первых поколений с небольшим объемом ОЗУ двоичная длина адреса была в несколько раз короче хранящегося в ячейке кода. Чтобы не уйти далеко в сторону, не будем разбираться в причинах данного соотношения, а продолжим рассмотрение изображенных на рис. 2.5 регистров.

Регистр команд служит для хранения исполняемой в данный момент команды программы. В ней не только задается действие, которое необходимо выполнить (например, сложение или логическая операция И), но также адреса ячеек памяти, с которыми требуемое действие необходимо произвести. Эти адреса по очереди копируются из регистра команд в регистр адреса ОЗУ, а затем данные считываются (или записываются) через регистр данных, как уже упоминалось выше. Остается только добавить, что регистр данных ОЗУ напрямую связан с АЛУ и может передавать последнему и принимать из него данные (чтобы не загромождать рисунок, эти связи не показаны). На-

деюсь, теперь общая картина выполнения процессором очередной инструкции читателям ясна: данные считываются из памяти, передаются в АЛУ, там обрабатываются и результат снова возвращается в ОЗУ через все тот же регистр данных.

Теперь перейдем наконец к главной цели нашего детального изучения — к реализации чтения информации с внешнего устройства (в нашем случае с перфоленты). Для реализации этого процесса служит еще один специальный регистр, куда и поступают данные от устройства. Обязательно обратите внимание на то, что на перфоленте одно машинное слово занимает несколько рядов (количество дорожек перфоленты заметно меньше разрядности машины). Следовательно, данные считываются по частям и постепенно сдвигаются. Только после того, как сформирована вся ячейка, ее содержимое будет переписано в регистр данных ОЗУ и лишь затем стандартным образом попадет в память. Во всей этой схеме отчетливо видно, что информация с внешнего устройства сначала попадает в процессор, формируется там и только тогда записывается в ОЗУ.

Разумеется, отдельные детали в описанном процессе могут у разных машин непринципиально отличаться, например, регистр адреса ОЗУ может являться частью регистра команд, а регистры данных для ОЗУ и внешних устройств даже совпадать. Однако сути дела это не меняет: в классическом варианте (напомним, что это первое и второе поколение вычислительной техники, а тогда машины по необходимости были "аскетически" простыми) вся информация поступает в машину через процессор.

2.2.3. Структура современных ЭВМ

Естественно, что в результате бурного развития технологии производства средств вычислительной техники классическая структура не могла не претерпеть определенных прогрессивных изменений. Начало этих изменений относится к третьему поколению ЭВМ.

Как известно, появление третьего поколения ЭВМ было обусловлено переходом от транзисторов к интегральным микросхемам (вспомните материал *разд. 1.3.1*). Значительные успехи в миниатюризации электронных схем не просто способствовали уменьшению размеров базовых функциональных узлов ЭВМ, но и создали предпосылки для заметного роста быстродействия процессора. Возникло существенное противоречие между высокой скоростью обработки информации внутри машины и медленной работой устройств ввода/вывода, в большинстве своем содержащих механические движущиеся части. Если бы процессор руководил работой внешних устройств по классической схеме (как описано выше), то значительную часть времени он был бы вынужден простаивать в ожидании информации "из внешнего мира", что существенно снижало бы эффективность работы всей ЭВМ в целом. Для решения этой проблемы возникла тенденция к освобождению цен-

трального процессора от функций обмена информацией и к передаче их специальным электронным схемам управления работой внешних устройств. Такие схемы имели различные названия: в ЕС ЭВМ это были каналы обмена [19], позднее — процессоры ввода/вывода или периферийные процессоры. Последнее время чаще всего используется термин *контроллер*[1] *внешнего устройства*. Наличие интеллектуальных контроллеров внешних устройств стало важной отличительной чертой машин третьего и в дальнейшем четвертого поколений.

Контроллер можно рассматривать как специализированный процессор, управляющий работой вверенного ему внешнего устройства по специальным встроенным программам обмена. Такой процессор имеет собственную систему команд. Например, контроллер накопителя на гибких магнитных дисках (дисковода) умеет позиционировать головку на нужную дорожку диска, читать или записывать сектор, форматировать дорожку и т. п. Сведения об успешности выполнения каждой такой операции заносятся во внутренние регистры контроллера, которые могут быть в дальнейшем прочитаны центральным процессором. Аналогичные контроллеры сейчас имеет любое внешнее устройство.

Высокий "интеллектуальный уровень" внешних устройств существенно изменяет идеологию обмена. Центральный процессор при необходимости произвести обмен выдает задание на его осуществление контроллеру, который создает канал между ОЗУ и внешним устройством. Дальнейшая передача информации протекает под руководством контроллера без использования аппаратно-программных средств центрального процессора. Последний получает возможность "заниматься своим делом", т. е. выполнять программу дальше (если по данной задаче до завершения обмена ничего делать нельзя, то можно в это время выполнять другие операции).

Опыт эксплуатации подтвердил полезность такой идеологии обмена, и при переходе к четвертому, микропроцессорному, поколению контроллеры внешних устройств получили дальнейшее развитие.

Мы не будем рисовать схему устройства ЭВМ третьего поколения. Лишь напомним читателю, что в то время вычислительная техника, в состав которой входили семейства машин IBM 360 и 370, а также "разработанные на их базе" ЕС ЭВМ, представляла собой сложные вычислительные комплексы коллективного пользования [33, 19]. Поэтому сразу перейдем к обсуждению внутренней структуры современной ЭВМ четвертого поколения, которая изображена на рис. 2.6. Из него видно, что для связи между отдельными функциональными узлами ЭВМ появилось принципиально новое устройство — общая шина (часто ее называют *магистралью*).

---

[1] Ударение на втором слоге: контро́ллер, а не контролёр, как иногда неправильно говорят.

Шина состоит из трех частей:

- шина данных, по которой передается информация;
- шина адреса, определяющая, куда именно передаются данные;
- шина управления, регулирующая процесс обмена информацией.

Примечание

Существовали модели компьютеров, у которых шины данных и адреса для экономии объединялись. У таких машин сначала на шину выставлялся адрес, а затем через некоторое время данные; для какой именно цели использовалась шина в данный момент, определялось сигналами на шине управления.

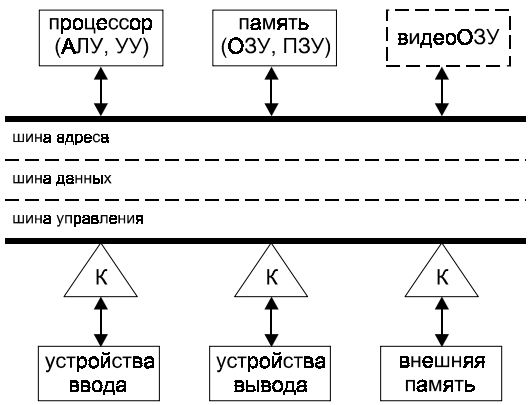


Рис. 2.6. Структура ЭВМ четвертого поколения

Как следует из изложенного выше теоретического материала, для согласования с шиной многочисленных внешних устройств, используются контроллеры, которые обозначены треугольником с буквой "К" внутри. Отметим, что хотя для каждого типа внешних устройств на схеме показан только один контроллер, это сделано исключительно ради компактности рисунка. На самом деле каждое устройство, например клавиатура и сканер, имеет собственный контроллер.

Описанную структуру легко пополнять новыми устройствами — это свойство называют *открытостью архитектуры*. Для пользователя открытая архитектура означает возможность свободно выбирать состав внешних устройств для своего компьютера, т. е. конфигурировать его в зависимости от круга решаемых задач.

На рис. 2.6 имеется еще один вид памяти, отсутствовавший в предыдущих поколениях ЭВМ — *видеоОЗУ*. Его появление связано с разработкой особого устройства вывода — дисплея (монитора). Основной частью дисплея служит электронно-лучевая трубка, которая отображает информацию примерно так

же, как это происходит в телевизоре; раньше для некоторых дешевых домашних моделей просто подключался обычный телевизор. Очевидно, что дисплей, не имея механически движущихся частей, является "очень быстрым" устройством отображения информации. Поэтому для ЭВМ третьего и четвертого поколений монитор является неотъемлемой частью.

Примечание

Строго говоря, дисплей был реализован уже на ЭВМ второго поколения МИР-2 — очень интересной во многих отношениях отечественной разработке. Красивое название ЭВМ на самом деле расшифровывалось весьма прозаически — Машина для Инженерных Расчетов; фотографию машины можно посмотреть в виртуальном музее по адресу [И-7].

Для получения на экране монитора стабильной картинки, ее надо где-то хранить. Для этого и существует видеопамять. Сначала содержимое видеопамяти формируется компьютером, а затем контроллер дисплея выводит изображение на экран.

Требуемый объем видеопамяти существенно зависит от характера информации (текст или графика) и от количества цветов изображения. Количество мегабайт видеопамяти в современном компьютере является одной из важных характеристик, влияющих на качество цветного изображения и скорость работы графической части.

Конструктивно видеоОЗУ может быть выполнено как часть обычного ОЗУ (так было в известном бытовом компьютере БК-0010), или содержаться непосредственно в контроллере дисплея (IBM PC) — именно поэтому на рисунке видеопамять показана пунктиром. Кстати, идея совмещения видеопамяти с обычным ОЗУ до сих пор жива. Она пережила второе рождение в связи с разработкой для дешевых моделей IBM PC так называемых интегрированных плат, на которых контроллеры внешних устройств монтируются непосредственно, а не подключаются к основной плате, как это обычно делается, в виде самостоятельных блоков. Существуют такие конструкции плат, у которых нет специального видеоОЗУ, а оно выделяется из общего блока памяти [21].

Собственную память могут иметь и другие узлы ЭВМ. Например, сетевая плата накапливает принимаемую по сети информацию в небольшом ОЗУ, а практически любой принтер также имеет собственную память (у некоторых лазерных принтеров она весьма значительна), где принятая информация хранится в ожидании печати.

Описывая магистральную структуру, мы упрощенно предполагали, что все устройства взаимодействуют через одну общую шину. С точки зрения понимания принципов этого вполне достаточно. Упомянем все же, что на практике такая структура применяется только для ЭВМ с небольшим количеством внешних устройств. При увеличении потоков информации между уст-

ройствами ЭВМ единственная магистраль перегружается, что существенно тормозит работу компьютера. Поэтому в состав современного компьютера может вводиться несколько дополнительных шин. Например, одна шина может использоваться для обмена с памятью, вторая — для связи с "быстрыми", а третья — с "медленными" внешними устройствами.

Отметим, что хорошее описание работы ЭВМ с магистральной архитектурой имеется в самом первом школьном учебнике по основам информатики и вычислительной техники [32].

2.2.4. Режим прямого доступа к памяти

Остановимся еще на одной важной особенности структуры современных ЭВМ. Поскольку процессор теперь перестал быть центром конструкции, стало возможным реализовывать по магистрали прямые связи между устройствами ЭВМ. На практике чаще всего используют передачу данных из внешних устройств в ОЗУ и наоборот. Режим, при котором внешнее устройство обменивается непосредственно с ОЗУ без участия центрального процессора, называется *прямым доступом к памяти* (ПДП). Для его реализации необходим специальный контроллер.

В режиме прямого доступа к памяти центральный процессор передает контроллеру ПДП необходимую для обмена информацию (режим обмена, начальный адрес ОЗУ, количество информации), после чего освобождается, а обменом руководит уже контроллер ПДП. Если такой обмен не задействует все возможности шины, то центральный процессор в это время может продолжать работу.

В качестве примера удачного использования режима ПДП в современном компьютере можно сослаться на работу звуковой карты. Поскольку воспроизведение музыки и звуков "с точки зрения процессора" происходит крайне медленно (сравните 500 МГц его тактовой частоты с максимальной частотой высококачественной дискретизации компакт-диска — 44 кГц!), то достаточно просто поместить в ОЗУ необходимые данные и сообщить контроллеру ПДП их адрес и количество. Последний вполне сможет самостоятельно и "неспешно" обеспечить передачу данных, которые требуются звуковой карте.

Несмотря на потенциальные возможности повышения производительности компьютера в данном режиме, не следует считать его настолько эффективным, чтобы отказаться от классического обмена с памятью под руководством и через процессор, о котором говорилось ранее. Так, по свидетельству П. Нортона, "поскольку дисковый ввод/вывод — сравнительно медленная операция, применение DMA[1] не очень сильно увеличивает общую производительность компьютера" [61]. В [15] также утверждается, что скорость об-

[1] По-английски режим ПДП называется Direct Memory Access (DMA).

мена ПДП ниже, чем у происходящего по программе центрального процессора.

Особо подчеркнем, что режим ПДП в машинах первого и второго поколения не существовал: изначально данные всегда принимались во внутренние регистры процессора и лишь затем попадали в ОЗУ, как это подробно было рассмотрено в *разд. 2.2.2.*

2.3. Выводы

Кратко сформулируем, что мы узнали из второй лекции.

- Подавляющее большинство вычислительных машин имеет одни и те же функциональные узлы. Если не обращать внимание на изменение терминологии, их состав для всех поколений ЭВМ, включая современное четвертое, примерно одинаков.
- Важнейшими функциональными узлами ЭВМ являются АЛУ и УУ, объединяемые сейчас в единое целое внутри микропроцессора, память различных видов, в том числе и внешняя, а также устройства ввода и вывода информации.
- Технологии изготовления узлов в ходе развития вычислительной техники несколько раз менялись революционным образом. Это привело к резкому уменьшению их размеров, существенному повышению быстродействия, а также появлению новых типов внешних устройств и существенному расширению их ассортимента.
- Взаимодействие между узлами ЭВМ за время развития ВТ было существенно усовершенствовано. От архитектуры, в которой единственный процессор являлся информационным и управляющим центром всей машины, произошел переход к магистральной архитектуре, при которой значительную часть работы по обмену информацией с внешними устройствами выполняют специализированные процессоры ввода/вывода — контроллеры.
- Не следует смешивать между собой два упомянутых выше типа архитектуры — классический и магистральный. (ЕС ЭВМ, принадлежащие к третьему поколению, имели некоторую особую переходную структуру; при первом знакомстве ее можно не рассматривать.)

2.4. Вопросы для осмысления

1. Что такое поколение ЭВМ и по каким признакам они различаются? Машины каких поколений вам приходилось видеть?
2. Перечислите все функциональные узлы ЭВМ и объясните назначение каждого. Какие из них находятся в системном блоке компьютера IBM PC?

3. Какие классические принципы до сих пор используются в современных ЭВМ?
4. Что такое адрес? Корректно ли описывать адрес стандартным 16-разрядным целым числом? (Указание: обратите внимание на отрицательные значения и ограниченный диапазон целых чисел в ЭВМ.)
5. Сравните взаимодействие узлов ЭВМ, построенных по классической и по магистральной схеме. Что общего и в чем различие?
6. Попробуйте самостоятельно дополнить схему рис. 2.5 регистрами АЛУ: сумматором и регистрами для хранения двух чисел. Проследите по полученной схеме за последовательностью выполнения сложения двух чисел в предположении, что команда в явном виде содержит в себе три адреса ОЗУ: адреса двух слагаемых и адрес результата.
7. Если вас заинтересовало устройство ЕС ЭВМ, о котором в лекции упомянуто довольно кратко, прочтите соответствующую литературу. Что такое селекторный и мультиплексный каналы и чем они отличаются? Можно ли в ЕС ЭВМ загрузить данные в ОЗУ, минуя центральный процессор?
8. Что такое контроллер? Подумайте, какими способами контроллер может, по-вашему, повысить производительность системы.
9. Оцените минимальный объем видеопамяти, требуемый для работы в режиме 24-битовой кодировки цвета при размере экрана 600×800.
10. Что такое режим ПДП? Почему для реализации его преимуществ требуется высокая пропускная способность шины?
11. Придумайте пример ситуации, когда центральный процессор не сможет продолжить выполнение программы до полного завершения процедуры ПДП. (Указание: последующая программа должна использовать вводимые данные.)

2.5. Любопытные эксперименты

Начиная с этой лекции появляется новый раздел. Поскольку он не совсем традиционный, кратко поясним его цели и содержание.

Разделы "Любопытные эксперименты" призваны на практике закрепить и расширить теоретические знания, которые излагались в лекции. Хотя данный материал не является обязательным и при чтении книги его можно пропустить, автор все же убедительно рекомендует с той или иной степенью подробности повторить описанные занимательные опыты на компьютере. Помимо достижения более глубокого понимания они достаточно поучительны и интересны сами по себе, а кроме того, могут служить некоторым образцом подхода к изучению поведения компьютера и его основных закономерностей.

Эксперименты довольно разнообразны по методам осуществления. В них используется как работа с некоторыми системными (служебными) программами, так и реализация небольших специально написанных программ. В качестве языка программирования выбран язык Паскаль как наиболее наглядный и подходящий для учебных целей. Все программы максимально короткие и тщательно проверялись, так что их повторение не должно вызывать особых трудностей у читателей. Можно также воспользоваться готовыми текстами программ или даже откомпилированными EXE-файлами, записанными на прилагаемом к книге CD-диске, но, по субъективному мнению автора, данный метод менее эффективен с точки зрения глубины усвоения материала.

Будет просто замечательно, если в результате выполнения описанных на страницах книги экспериментов читатель придумает и проделает свои собственные. Автор будет благодарен, если вы пришлете их описание по электронной почте.

2.5.1. Представление чисел в машине

В компьютерной литературе утверждается, что компьютер хранит и обрабатывает данные в двоичной системе. А действительно ли это так? Мы вводим десятичные числа и получаем на экране десятичный результат. Можно ли быть уверенным в том, что компьютер производил действия именно в двоичной системе? Как это проверить?

Цель эксперимента. Убедиться, что внутреннее представление чисел в современном компьютере по-прежнему двоичное.

Наберем следующую несложную программу, написанную на языке Паскаль (листинг 2.1).

Листинг 2.1. Программа для проверки внутреннего представления чисел в компьютере

```
PROGRAM BinaryRepresentation1;
VAR   m,m0,i: INTEGER; x,h: REAL;
BEGIN m0:=128;
      FOR m:=m0-1 TO m0+1 DO
          BEGIN h:=1/m; x:=0;
                FOR i:=1 TO m DO x:=x+h;
                WRITELN(m,'  ', x-1)
          END
END.
```

Для тех, кто не очень хорошо знаком с языком Паскаль, несколько комментариев. Внешний цикл `FOR` обеспечивает троекратное исполнение програм-

мы для значений m от m0 - 1 до m0 + 1 (в нашем случае это 127, 128 и 129). Нетрудно заметить, что речь идет о числе, являющемся точной степенью двойки (т. е. 128) и двух "соседних". Далее для каждого из трех чисел m раз проделывается суммирование величины 1/m и печатается разность, показывающая, насколько полученное число отличается от 1. Запустив программу, мы увидим на экране дисплея следующий результат:

```
127 -7.2759576142E-12
128  0.0000000000E+00
129  2.1827872843E-11
```

Иными словами, только для числа 1/128, которое представляется в двоичной системе точно, результат тождественно равен нулю. А вот для двух других чисел результат получился хоть и очень маленький (10^{-11} — 10^{-12}), но ненулевой.

Может быть это случайность? Проверим для других m0, например 512 или 2048 — результат будет аналогичный. В то же время, при m0 = 126, все три результата будут отличны от нуля, т. е. степени двойки действительно являются "особенными".

Отметим, что данный эксперимент можно провести и для других языков программирования. Например, в [25] описаны аналогичные результаты для языка Basic.

Вывод. Полученные результаты можно объяснить единственным способом: компьютер действительно хранит числа и производит вычисления в двоичной системе. Именно поэтому только для степеней двойки ему удается получить точный результат.

2.5.2. Двоичное представление чисел в памяти

Можно ли заглянуть в память компьютера, чтобы увидеть, каким образом там хранится информация? Опишем еще один достаточно убедительный эксперимент, подтверждающий, что до сих пор информация в компьютере хранится именно в двоичной системе.

Цель эксперимента. Проверить утверждение о том, что числа в памяти компьютера хранятся в виде двоичного кода.

Попробуем извлечь целое число непосредственно из ОЗУ и проверить, действительно ли там находится тот самый двоичный код, о котором столько пишут в книгах по информатике.

Для проверки потребуется очень небольшая, но нетривиальная программа на языке Паскаль (листинг 2.2). Как и в предыдущем случае, можете провести эксперимент с другим языком.

Листинг 2.2. Программа для проверки хранения чисел в виде двоичного кода

```
PROGRAM BinaryRepresentation2;
VAR   i,m,b: BYTE;
BEGIN b:=65;
      m:=$80; {bin = 1000 0000}
      FOR i:=1 TO 8 DO
          BEGIN IF (MEM[SEG(b):OFS(b)] AND m) = 0
                   THEN WRITE('0')
                   ELSE WRITE('1');
                m:=m SHR 1
          END; WRITELN
END.
```

Кратко разберемся, как она работает. В "изучаемую" переменную `b` поместим некоторое значение, например, 65. Затем, предварительно подготовив в переменной `m` так называемую "маску" для выделения самого старшего бита числа, выведем на экран по очереди содержимое всех 8 бит переменной `b`. Для этого в цикле будем извлекать из памяти байт, в котором лежит значение переменной `b` (в программе это записано как довольно специфическое выражение `MEM[SEG(b):OFS(b)]`, которое, поверьте автору на слово, действительно "напрямую" извлекает из ОЗУ значение "того самого" байта), выделять из него один бит с помощью операции `AND` и выводить результат анализа на экран. Подчеркнем, что последняя строка цикла сдвигает "маску" на один разряд вправо, тем самым в следующий раз будет выделяться соседний бит.

Запустив описанную программу, увидим на экране строку

01000001

которая и является двоичным представлением исходного числа 65. Разумеется, можно повторить опыт и с другими значениями. Это читатели без труда могут проделать самостоятельно.

Вывод. В памяти компьютера числа хранятся в двоичном виде.

2.5.3. Двоичное представление информации на диске

Надеюсь, после первых двух экспериментов все читатели убедились, что данные внутри компьютера действительно хранятся в двоичном виде. А что происходит при записи данных во внешнюю память, например, на магнитный диск? Для этого проведем новый эксперимент.

Цель эксперимента. Убедиться, что данные в файлах представлены в двоичном виде. Проверить утверждение о принципиальной неразличимости двоичных кодов для разных видов информации.

Рассмотрим еще один эксперимент, представляющий собой некоторое усовершенствование предыдущего опыта. Он показывает, что двоичное представление информации хранится не только в ОЗУ, но и в файлах на дисках.

Начнем с того, что в простом текстовом редакторе, таком как Блокнот, наберем единственный символ — латинскую заглавную букву A. Ничего больше не нажимая на клавиатуре (даже клавишу <Enter>!), сохраним этот простейший текстовый файл из одной буквы под именем proba.txt.

Проверим, что получилось, используя широко распространенную программу Far (аналог Norton Commander для среды Windows). Используя для просмотра полученного файла традиционную клавишу <F3>, дополнительно переведем программу в режим Hex-кодов, нажав клавишу <F4>, и увидим следующую картину (рис. 2.7).

Рис. 2.7. Просмотр пробного текстового файла с помощью программы Far

Отчетливо видно, что текст состоит из единственного символа A (справа), имеющего шестнадцатеричный код 41 (слева). Попутно обратим внимание на выделенный на рисунке размер файла — 1 байт.

Итак, первый вывод, который можно сделать уже сейчас, — символы в файлах на внешних устройствах хранятся в виде их числовых кодов. Но мы пойдем дальше. Наберем еще одну программу на Паскале, которая во многом совпадает с предыдущей (листинг 2.3).

Листинг 2.3. Программа проверки двоичного представления данных на диске

```
PROGRAM BinaryRepresentation3;
VAR   b,i,m: BYTE; f: FILE OF BYTE;
BEGIN ASSIGN(f,'proba.txt'); RESET(f);
      READ(f,b);
      m:=$80; {bin = 1000 0000}
      FOR i:=1 TO 8 DO
          BEGIN IF (b AND m) = 0
                    THEN WRITE('0')
                    ELSE WRITE('1');
                m:= m SHR 1
          END;
      WRITELN; WRITELN(b);
      CLOSE(f);
END.
```

Приведенная выше программа читает из файла в переменную b значение его единственного байта, а затем выводит его на экран в двоичном виде (алгоритм вывода полностью заимствован из предыдущего примера), а также в десятичном.

В результате исполнения программы на экране появятся две строки:

```
01000001
65
```

т. е. значение числа в двоичной и десятичной системах соответственно.

Выводы. Непосредственный вывод данных из файла продемонстрировал, что не только в ОЗУ, но и на диске данные хранятся в двоичном виде. Особо подчеркнем, что данный эксперимент дополнительно показал еще одно интересное свойство хранения информации в современных компьютерах — по виду двоичного кода принципиально невозможно определить, является ли он числом, текстом или чем-то другим. Действительно, мы сохранили текст из единственной буквы, а прочитали из файла целое число!

2.5.4.* Может ли машина сама формировать себе программу?

Символом * договоримся помечать наиболее сложные эксперименты. Данный эксперимент как раз является таковым.

Из теории следует, что программа, как и данные, является двоичным кодом. Поэтому естественно предположить, что команды программы можно формировать или изменять программным способом. Так ли это? Конечно так, и те, кто хоть раз скомпилировал программу этим уже пользовался! Ибо что такое компиляция, если не формирование программы на языке процессора с помощью специализированной программы-компилятора? Но это тоже теория. А как обстоит дело на практике?

Цель эксперимента. Показать, что машина способна внести внутрь своей программы некоторую команду и затем исполнить ее. Продемонстрировать еще раз универсальность двоичного кода и, как следствие, невозможность по виду кода установить, является ли он программой или данными.

Итак, уважаемые читатели, на очереди очень интересный и достаточно глубокий эксперимент. Страшно сказать — мы реализуем самоформирующуюся программу на ассемблере! Но не пугайтесь, пожалуйста, все будет достаточно просто и успешно, если внимательно прочесть подробный текст и четко следовать не менее подробным инструкциям. Тем немногочисленным читателям, которые заинтересуются ассемблером и захотят познакомиться с ним подробнее, автор убедительно рекомендует замечательную книгу Питера

Нортона с Джоном Соухэ [62], которую можно начинать читать безо всякой предварительной подготовки.

Примечание

Возможно, если вам захочется разобраться с данным примером чуть глубже, стоит реализовать его после чтения *лекции 5*, когда станут лучше понятны детали работы ОЗУ и процессора. Тем не менее, следуя подробным инструкциям в тексте примера, его можно выполнить уже сейчас. Автор считает это вполне возможным по той простой причине, что все равно далеко не все тонкости устройства процессора Intel будут рассмотрены в данной книге, а два или четыре факта принимать на веру — это не очень существенно.

Итак, рассмотрим программу, которая ни много ни мало сама сформирует внутри себя логическую команду и выполнит ее. Начнем с того, что выпишем из любого справочника по системе команд IBM PC коды логических операций между двумя регистрами микропроцессора — `AX` и `BX` (табл. 2.1).

Таблица 2.1. *Коды логических операций между регистрами* `AX` *и* `BX`

Команда	Код	Пояснение
`AND AX,BX`	21 D8	Логическое И
`OR AX,BX`	09 D8	Логическое ИЛИ
`XOR AX,BX`	31 D8	Исключающее ИЛИ

Код формируемой команды (возьмем для определенности действий первый — `21 D8`) договоримся поместить сначала в регистр `CX`, а затем оттуда записать в память.

Напишем на листочке бумаги следующую программу и разберем ее (табл. 2.2).

Таблица 2.2. *Программа (с пояснениями), формирующая и выполняющая логическую команду*

Адрес	Команда	Действие	Пояснение
100	`mov ax,1234`	Поместить `1234` в регистр `AX`	1-й операнд
103	`mov bx,F0F0`	Поместить `F0F0` в регистр `BX`	2-й операнд
106	`mov cx,D821`	Поместить `D821` в регистр `CX`	Команда И
109	`mov [10D],cx`	Скопировать `CX` в ячейку `10D`	Записать команду в ОЗУ
10D	???	[Сюда запишется команда из `CX`]	Логическая операция

Итак, первые две команды задают абсолютно произвольные константы (все данные в таблице записаны в шестнадцатеричном виде), над которыми будет произведена логическая операция, в регистры AX и BX. Следующая заносит в CX "перевернутый" код операции (D8 21 вместо 21 D8). Причину этой странности мы разберем позднее, в *разд. 4.2.2*, когда будем говорить об организации памяти ЭВМ и хранении в ней многобайтовых данных. А пока давайте отнесемся к ней как к "научно-медицинскому факту", т. е., выражаясь более серьезно, примем его без доказательства. Наконец, последняя из написанной серии инструкций наиболее важная: она заносит в ячейку со следующей командой (ее адрес 10D) выбранную логическую операцию. Следуя дальше по программе, процессор выполнит ее, подтверждая тем самым не совсем обычный тезис о том, что компьютер способен сам себе формировать программу.

Надеюсь, уважаемые читатели, вы внимательно прочли предыдущий абзац, заглядывая при этом в таблицу, и разобрались в сути нашей короткой программки. Теперь остается проверить ее на компьютере. Для этого мы используем входящий в состав ОС Windows стандартный отладчик DEBUG. Чтобы его запустить, щелкните по кнопке **Пуск** и выберите пункт **Выполнить**. При этом на экране возникнет диалоговое окно, в котором вы должны набрать с клавиатуры имя требуемой вам программы (рис. 2.8).

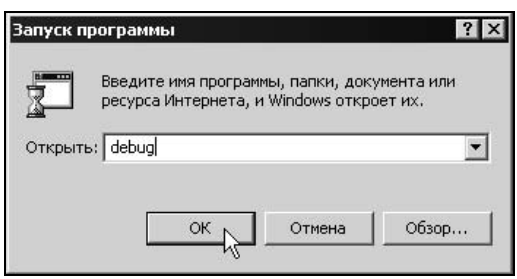


Рис. 2.8. Вызов программы Debug

После этого появится пустое черное окно, в верхней строчке которого будет высвечен символ "–", который является признаком готовности отладчика принять команду. Чтобы набрать нашу программу, введем короткую команду a (ассемблер) и нажмем клавишу <Enter>. После этого очень внимательно наберем 4 команды mov из второго столбца таблицы.

Примечание

При вводе пробелы ставятся только после названия инструкции mov, все остальное набирается слитно.

Вместо следующей строки сразу нажмем клавишу <Enter>, и Debug поймет, что ввод программы уже закончен. Снова появится традиционное приглашение к диалогу в виде "минуса". Для проверки правильности набора введем

`u` и снова нажмем клавишу <Enter>. В ответ на это Debug выведет находящуюся в его памяти программу (рис. 2.9).

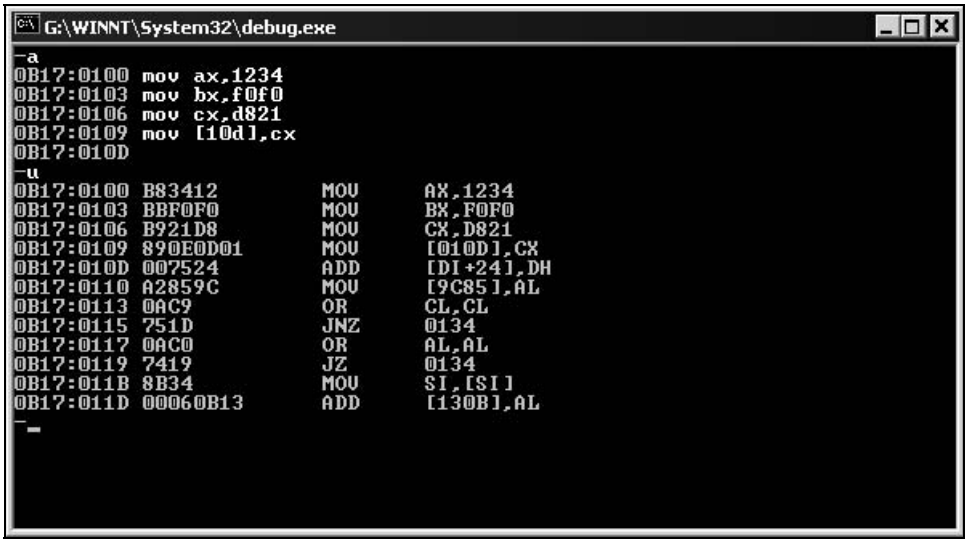


Рис. 2.9. Ввод программы и его проверка

Внимательно проверьте ее и в случае ошибки лучше повторите все с самого начала, хотя, разумеется, при лучшем знании отладчика ошибку не очень сложно исправить.

Обратите внимание на следующий важный факт. Начиная с ячейки `10D` информация не вводилась, поэтому то, что там будет, достаточно непредсказуемо (на экране вы, скорее всего, увидите другой текст). Для нас сейчас особенно важно, что интересующая нас ячейка `10D` перед запуском программы не определена.

Теперь, когда все набрано и проверено, исполним нашу программу. Для этого командой `t5` попросим Debug выполнить очередные 5 команд (как вы, конечно, помните, мы набрали только четыре, но очень надеемся, что пятую компьютер занесет в память сам). Если ошибок не было, на экране появится текст, аналогичный показанному на рис. 2.10.

После каждой команды Debug вывел подробную информацию о состоянии процессора.

Примечание

Подобный режим, когда после каждой выполненной команды машина выдает подробную информацию о результатах своей работы, программисты называют *трассировкой* (не случайно в качестве обозначения команды мы использовали латинскую букву "t").

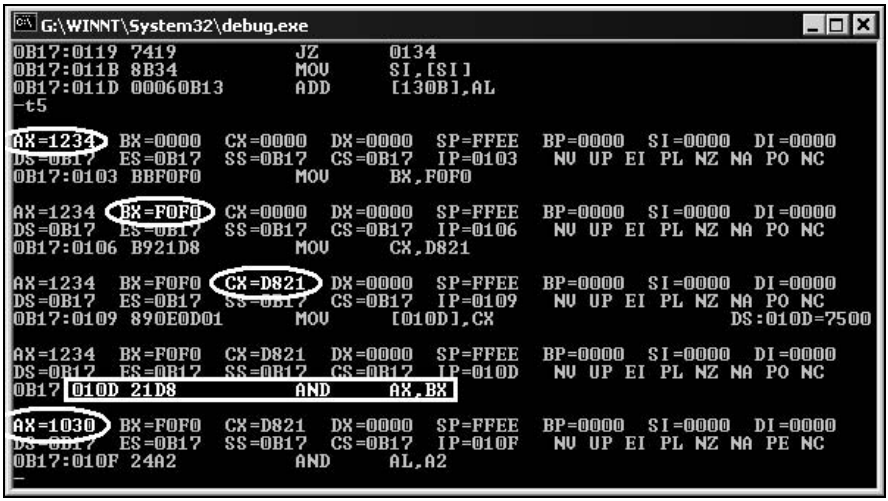


Рис. 2.10. Запуск и исполнение программы

Для удобства обсуждения, наиболее важные фрагменты на рис. 2.10 выделены. Отчетливо видно, как первые три команды постепенно заполняют значения исходных регистров, после четвертой в память заносится выбранная нами команда `AND AX,BX`, о чем свидетельствует последняя строчка вывода. Наконец, "свежезанесенная" в ОЗУ операция (пятая по счету) исполняется, и работа программы прекращается. Результатом ее выполнения является значение регистра `AX`, которое, как видно из рисунка, равняется 1030. Нетрудно убедиться, что ответ правильный (рис. 2.11).

$$a = 1234_{16} = 0001\ 0010\ 0011\ 0100_2$$
$$b = F0F0_{16} = 1111\ 0000\ 1111\ 0000_2$$
$$a\ AND\ b = 1030_{16} = 0001\ 0000\ 0011\ 0000_2$$

Рис. 2.11. Проверка результата выполнения программы

Если вам еще не надоели наши упражнения, замените содержимое регистра `CX` на `D8 09 (OR)` или `D8 31 (XOR)` и повторите эксперименты. Правильные ответы должны быть `F2F4` и `E2C4`, в справедливости чего предлагаю читателям убедиться самостоятельно.

Вывод. Таким образом, мы своими глазами увидели, что, следуя придуманной нами программе, машина смогла вписать в нее новую (отсутствовавшую при старте!) команду и выполнить ее. Данный эксперимент подтверждает и другое важное теоретическое положение о том, что не существует принципиального различия между данными и кодом программы: команда программы может при определенных условиях служить данными для другой команды.

Некоторые читатели, склонные к глобальным обобщениям, могут рассматривать данный любопытный эксперимент как повод для раздумий над традиционным философским вопросом: может ли машина мыслить?

Между прочим, помимо положительной (созидательной) функции, способность программы менять саму себя может играть и негативную (разрушительную) роль. Например, если в результате неправильного обращения к сложной структуре данных выйти за ее пределы, то компьютер запишет результат вне структуры, в том числе, возможно, и в другую часть программы. В последнем случае результат выполнения испорченного фрагмента окажется достаточно непредсказуемым. Поверьте опыту автора, подобные ситуации совсем не являются экзотикой, с которой встречаются только компьютерные профессионалы; напротив, они чаще всего наблюдаются именно у неопытных программистов.

2.5.5. Можно ли обойтись без умножения?

Конструкторы заложили в фундамент компьютера двоичную систему по двум причинам: легко реализовать функциональные элементы с двумя устойчивыми состояниями и правила действий над двоичными данными наиболее простые. Первая причина явно выходит за рамки нашего курса, а вот о второй давайте поговорим на примере арифметических действий.

Из четырех известных каждому школьнику действий арифметики ЭВМ обычно, как ни странно, непосредственно умеет выполнять только одно — сложение. Вычитание, умножение и деление могут быть тем или иным способом сведены к сложению в сочетании с другими простыми действиями, такими как, например, сравнение, логическое НЕ (инвертирование) и сдвиг. Мы рассмотрим операцию умножения, поскольку описание того, как преобразовать вычитание к сложению, легко найти в литературе (см. [3, 47]).

> **Цель эксперимента.** Доказать, что умножение для компьютера не является обязательной операцией, поскольку в двоичной системе его можно реализовать в виде программы из более простых операций.

Для получения алгоритма умножения двоичных чисел вспомним школьные правила умножения столбиком: каждое последующее произведение записывается со сдвигом на один разряд влево и прибавляется к результату (обычно при традиционном способе все сложения откладывают "на потом", но с таким же успехом можно суммировать и по ходу процесса). Кроме того, немаловажно, что при перемножении двоичных чисел произведение либо обращается в ноль, если текущая цифра множителя равна нулю, либо повторяет множимое, когда она является единицей. Учитывая все сказанное, получаем следующий несложный алгоритм перемножения для двух целых положительных чисел *a* и *b* (рис. 2.12).

Остается только добавить, что `SHL` — это сдвиг множимого на один разряд влево, `SHR` — вправо, а логическая операция `AND` с константой 1 позволяет

выделить младший бит множителя с целью формирования очередного произведения.

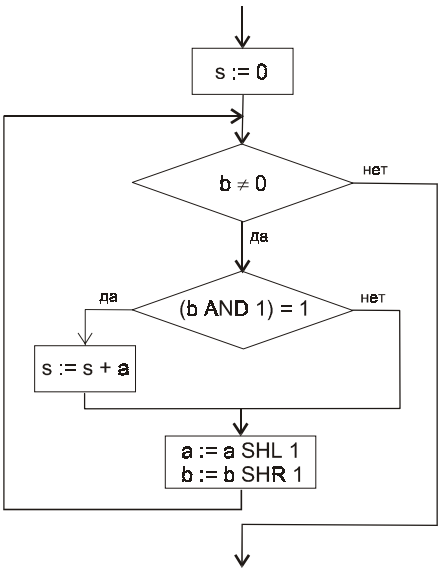


Рис. 2.12. Блок-схема алгоритма двоичного умножения

В заключение приведем для примера решение рассматриваемой задачи на языке Паскаль. Оно выглядит достаточно просто (листинг 2.4).

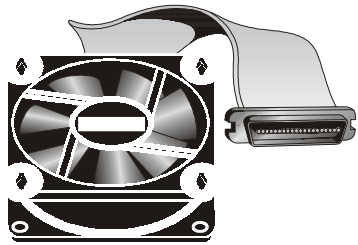
Листинг 2.4. Программа, реализующая двоичное умножение целых чисел

```
PROGRAM Multiplication;
VAR   a,b,s: INTEGER;
BEGIN WRITE('a='); READLN(a);
      WRITE('b='); READLN(b);
      s:=0;
      WHILE b<>0 DO
            BEGIN IF (b AND 1) = 1
                     THEN s:=s+a;
                  a:=a SHL 1; b:=b SHR 1
            END;
      WRITELN('a*b=', s)
END.
```

Запустите его и убедитесь, что, пользуясь описанным алгоритмом, программа действительно перемножает числа.

Вывод. Благодаря замечательной простоте правил арифметических действий в двоичной системе счисления, минимальный набор инструкций процессора может быть существенно уменьшен.

ЛЕКЦИЯ 3



Процессор ЭВМ

В прошлой лекции мы познакомились с функциональными блоками ЭВМ и основными принципами их взаимодействия. Теперь, когда общая картина нам ясна, вполне разумно подробнее изучить каждый из узлов в отдельности.

Начнем мы, естественно, с процессора. Именно вопросам его устройства и работы будет посвящена данная лекция.

3.1. Назначение процессора и его устройство

Процессор — это блок ЭВМ, предназначенный для автоматического считывания команд программы, их расшифровки и выполнения. Термин, по-видимому, наиболее близко приближается к английскому глаголу "process", имеющему один из вариантов перевода "обрабатывать". И действительно, основное назначение процессора состоит в автоматической обработке информации по заданной программе.

Примечание

Термин "процессор" часто используется при классификации программного обеспечения — текстовой процессор, табличный процессор и т. п. Очевидно, что к теме настоящей лекции такое использование слова "процессор" отношения не имеет, поэтому здесь мы не будем касаться этого значения.

Будучи центральным устройством ЭВМ, процессор во многом определяет ее возможности и производительность. Не случайно при характеристике компьютера тип процессора всегда указывается в первую очередь. Тем не менее не следует забывать, что общая эффективность вычислительной системы зависит не только от процессора, но и от остальных ее компонентов, даже от их согласования между собой. Иными словами, "медленная" память или

устаревшая графическая часть всегда могут свести на нет все преимущества новейшего процессора.

Мы уже знаем из *разд. 2.1*, что традиционными главными частями процессора являются арифметико-логическое устройство (АЛУ) и устройство управления (УУ). В данной лекции мы также ограничимся таким делением, хотя при более детальном рассмотрении иногда приводятся некоторые дополнительные части (см. [33]).

Но сначала остановимся на одном терминологическом моменте.

3.1.1. Процессор и микропроцессор

Первоначально процессор изготовлялся из отдельных деталей: транзисторов, микросхем невысокого уровня интеграции (с небольшим числом элементов) и т. д.; его размеры могли быть весьма значительными. Пожалуй, наиболее впечатляюще выглядел процессор ЭВМ серии ЕС (Единая Система ЭВМ относится к третьему поколению). Он представлял собой отдельный весьма внушительный шкаф, габариты которого можно представить себе из приводимой на рис. 3.1 фотографии: процессор на ней находится слева непосредственно за спиной оператора, его ширина и высота соизмеримы с человеком, а толщина — в несколько раз больше.

Рис. 3.1. Общий вид ЕС ЭВМ; ее процессор представлял собой шкаф весьма внушительных размеров

Прогресс в области микроэлектроники привел к тому, что весь процессор удалось разместить внутри одного кристалла. Таким образом, он стал отдельной самостоятельной микросхемой и получил новое название — *микропроцессор*. На рис. 3.2 изображен один из современных микропроцессоров.

Удерживающие его пальцы дают хорошее представление о габаритах изделия.

Рис. 3.2. Так выглядит типичный современный микропроцессор

Примечание

Следует иметь в виду, что размер корпуса в основном определяется необходимостью разместить несколько сот металлических выводов, с помощью которых процессор присоединяется к плате; собственно кристалл значительно меньше и имеет размер порядка 1 см.

То, что процессор удалось поместить внутри одной микросхемы, не просто уменьшило размеры этого узла, но и создало предпосылки для существенного увеличения скорости работы процессора (благодаря сокращению длины соединений) и повышения его надежности (за счет отсутствия внешних проводников). К сожалению, есть и отрицательные последствия: уменьшение габаритов процессора приводит к ухудшению условий теплоотдачи от микроскопических его элементов, что существенно повышает требования к теплоотводящим свойствам конструкции. Не случайно для охлаждения современных процессоров, особенно работающих на очень высоких частотах, обязательно используются металлические радиаторы достаточно большой площади и всевозможные вентиляторы (часто их жаргонно называют "кулерами" — т. е. по-русски "охладителями", "холодильниками"). Последние модели микропроцессоров даже содержат внутри кристалла специальные датчики температуры и поэтому способны автоматически выключаться, предохраняя себя от перегрева.

Интересно отметить, что первоначально микропроцессоры создавались не столько ради нужд вычислительной техники, сколько по другим технологическим причинам. Дело в том, что существенно уменьшить стоимость производства микросхем можно только при их массовом выпуске, а для этого изделия должны быть максимально универсальными. Поэтому-то и возникла идея производить универсальную микросхему, которая настраивается на конкретные задачи с помощью программирования (иногда в книгах это называют заменой аппаратной логики на программную). И только когда про-

изводство таких программируемых микросхем достигло определенного уровня, появились микропроцессоры в полном смысле этого слова, представляющие собой миниатюрный процессор вычислительной машины. Если читателей заинтересовала история развития производства микропроцессоров, то им можно порекомендовать популярные материалы [10, 78], которые содержат много интересных сведений по данному вопросу.

Сегодня развитие микропроцессоров требуется не только для вычислительной техники, где они используются в качестве центрального процессора и для управления сложными периферийными устройствами. Все большее число таких микросхем ставится в изделия, напрямую не связанные с ЭВМ, в том числе и бытовые: лазерные аудио- и видеопроигрыватели, телетекст и пейджинговая связь, программируемые микроволновые печи и стиральные машины, и многое-многое другое. Очевидно, что количество таких управляемых микропроцессорами устройств будет все время возрастать.

3.1.2. Арифметико-логическое устройство

Данная часть процессора служит для реализации всех тех операций, которые умеет выполнять процессор. Именно здесь осуществляются все арифметические действия, а также логические операции, сравнение данных, сдвиги и т. д. Полный ассортимент инструкций ЭВМ будет более подробно обсуждаться несколько позднее в *разд. 3.3*.

Чтобы составить для себя некоторое представление об устройстве АЛУ, рассмотрим логику выполнения сложения двух чисел.

Перед выполнением данной операции, прежде всего, необходимо куда-то поместить оба слагаемых. Для этого в процессоре имеются специальные схемы, которые называются регистрами. *Регистр* — это типовой узел ЭВМ, предназначенный для временного хранения данных или выполнения над ними некоторых действий. Широко распространены, например, так называемые *сдвиговые регистры*, способные сдвинуть набор нулей и единиц влево или вправо; для чего такие действия могут потребоваться, мы уже видели в *разд. 2.5.5*. Хочется подчеркнуть, что каждый отдельно взятый бит регистра представляет собой электронное устройство, с которым мы уже немного знакомы по первой лекции, — это триггер *(см. разд. 1.3.1)*.

Итак, в специальные регистры процессора помещены оба слагаемых. Для осуществления процесса суммирования требуется еще одно устройство, называемое *сумматором*. В него копируется первое слагаемое, а затем прибавляется второе, причем результат, естественно, тоже получается в сумматоре. Теперь его можно либо записать в память, либо использовать для дальнейших вычислений.

Важной функцией АЛУ является анализ полученного после выполнения команды результата. Обычно проверяется два свойства: равенство или нера-

венство нулю (устанавливается факт совпадения всех разрядов сумматора с нулем или отсутствие такового) и отрицательность или неотрицательность ответа (для этого достаточно скопировать знаковый разряд числа, который всегда установлен в единицу для отрицательных чисел и сброшен в ноль во всех остальных случаях). Читатель легко может самостоятельно убедиться, что этих двух бинарных (двоичных) признаков вполне хватает, чтобы скомбинировать любое из шести математических соотношений неравенства двух чисел. Результаты описанного анализа сохраняются в виде отдельных битов в особом регистре, который имеет в разных моделях процессоров довольно разнообразные названия: регистр флагов, регистр признаков или регистр состояния. Данные этого регистра могут быть в дальнейшем использованы при реализации команд условных переходов.

Вопрос о том, как выполняются остальные арифметические операции, мы уже рассматривали *(см. разд. 2.5.5)*. Подчеркнем только и запомним на будущее, что операции умножения и деления являются значительно более трудоемкими, чем сложение и вычитание.

Обратите внимание на то, что АЛУ способно лишь принять извне готовые числа и выставить в сумматоре результат действия над ними. Для того чтобы найти нужные числа и сохранить результат, в процессоре существует другое устройство — устройство управления.

3.1.3. Устройство управления

Итак, чтобы обеспечить автоматические вычисления по программе, процессор должен уметь выполнять еще целый ряд дополнительных действий:

- ❒ извлекать из памяти очередную команду;
- ❒ расшифровывать ее и преобразовывать в последовательность стандартных элементарных действий;
- ❒ заносить в АЛУ исходные данные;
- ❒ сохранять полученный в АЛУ результат;
- ❒ обеспечивать синхронную работу всех узлов машины.

Для выполнения всех этих функций и служит устройство управления.

Как и АЛУ, УУ содержит несколько важных регистров для хранения информации, необходимой в ходе выполнения текущей команды. Наиболее важными из них являются счетчик (регистр) адреса очередной команды программы и регистр команд, в котором хранится код выполняемой в данный момент операции.

Анализу последовательности действий, которые производит процессор при выполнении каждой команды, посвящен *разд. 3.2*.

3.1.4. Программно-доступные регистры

Описанные выше служебные регистры АЛУ и УУ, используемые процессором для обеспечения своего функционирования, как правило, недоступны программисту. Однако почти все процессоры, кроме самых старых и примитивных, имеют некоторый набор дополнительных регистров, предназначенных для использования программным обеспечением. Их часто называют *регистрами общего назначения* (РОН), подчеркивая тем самым универсальность их функций. В РОН может храниться как непосредственно обрабатываемая информация (числа, коды символов и т. п.), так и ссылки на те ячейки памяти, где находится такая информация. Обсуждение последней возможности придется отложить до следующей лекции, где мы будем говорить об организации памяти ЭВМ.

Количество регистров и их устройство в различных процессорах различно. В некоторых из них — таких, как, например, в замечательном по своей наглядности и логичности процессоре семейства машин PDP (Programmed Data Processor — программно-управляемый процессор данных) (в нашей стране такая же система команд использовалась в машинах серии ДВК, бытовых компьютерах семейства БК и школьном КУВТ УКНЦ), все регистры абсолютно равноправны. В других, наоборот, каждый регистр обладает определенными уникальными свойствами, что не позволяет использовать вместо него какой-либо другой. К указанному типу относятся все процессоры семейства Intel, что делает их очень неудобными для изучения.

Чтобы читатель получил некоторое первоначальное представление о работе с регистрами процессора, проиллюстрируем сказанное примером для процессора Intel. Рассмотрим команду, которая состоит из двух байт с двоичными кодами 0000 0001 и 1100 1000 и в символических обозначениях записывается

```
ADD AX, CX
```

Эта инструкция заставит процессор сложить числа, хранящиеся в регистрах `AX` и `CX`, а результат занесет в `AX`. Характерно, что регистр `AX` в такой команде является фиксированным и не может быть заменен, зато вместо `CX` можно поставить другой регистр: `BX` или `DX`. Естественно, код команды в последних случаях несколько изменится.

3.1.5. Разрядность процессора

Под *разрядностью* процессора обычно понимают число одновременно обрабатываемых им битов. Формально эта величина есть количество двоичных разрядов в регистрах процессора и для наиболее распространенных современных моделей она равна 32. Тем не менее, все не так просто. Дело в том, что помимо описанной внутренней разрядности процессора существует еще разрядность шины данных, которой он управляет, и разрядность шины ад-

реса *(см. разд. 2.2.3)*. Разрядность регистров и разрядность шины данных влияют на длину обрабатываемых данных, а вот разрядность шины адреса R определяет максимальный объем памяти, который способен поддерживать процессор. Эту характеристику часто называют величиной *адресного пространства*, и она может быть вычислена по простой формуле 2^R. Действительно, R двоичных разрядов позволяют получить именно такое количество неповторяющихся чисел, т. е. в нашем случае, адресов памяти.

Три описанные выше характеристики разрядности не всегда совпадают (табл. 3.1). Данные для таблицы взяты из книги [14].

Таблица 3.1. *Данные о разрядности некоторых процессоров*

Процессор	Разрядность			Объем памяти
	регистров	шины данных	шины адреса	
Intel 8086	16	16	20	до 1 Мб
Intel 80286	16	16	24	до 16 Мб
Intel 80386	32	16	24	до 16 Мб
Intel 80486	32	32	32	до 4 Гб
Pentium	32	64	32	до 4 Гб
Pentium II	32	64	36	до 64 Гб

Из табл. 3.1, в частности, видно, что для 386-го процессора все три разрядности различны.

3.2. Как работает процессор

3.2.1. Основной алгоритм работы процессора

В данном разделе рассмотрена последовательность действий при выполнении очередной команды в ЭВМ. Сравнение показывает, что рабочий цикл по своему содержанию в основном одинаков для всех вычислительных машин различных поколений и отличается лишь некоторыми техническими деталями реализации.

Как уже отмечалось в *разд. 2.2.1*, важной составной частью фон-неймановской архитектуры является счетчик адреса команд. Он постоянно указывает на ячейку памяти, в которой хранится следующая команда программы. Считав очередную команду из памяти, процессор сразу же увеличивает значение счетчика так, чтобы он показывал на следующую команду. Затем считанная команда расшифровывается и выполняется (рис. 3.3).

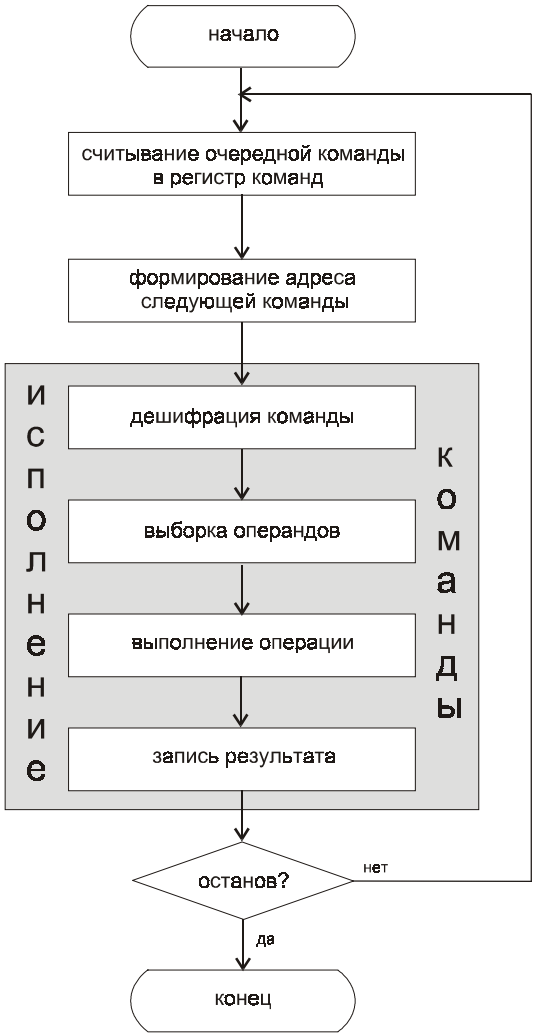


Рис. 3.3. Основной алгоритм работы процессора

Таким образом, ЭВМ непрерывно выполняет те или иные программы, причем каждая из них в свою очередь может загружать новые программы.

При выполнении каждой команды вычислительная машина проделывает определенные стандартные действия, описанные ниже.

1. Согласно содержимому счетчика адреса команд считывается очередная команда программы. Ее код заносится на хранение в регистр команд.
2. Счетчик команд автоматически изменяется так, чтобы в нем содержался адрес следующей команды. В простейшем случае для этой цели доста-

точно к текущему значению счетчика прибавить некоторую константу, определяющуюся длиной команды.

3. Считанная в регистр команд операция расшифровывается, извлекаются необходимые данные, над ними в АЛУ выполняются требуемые действия и, если предусмотрено операцией, результат записывается в ОЗУ.

   Затем во всех случаях, за исключением команды останова, описанные действия циклически повторяются.

Примечание

Данный алгоритм должен содержать еще один этап, связанный с анализом прерываний; о нем пойдет речь в *лекции 6*, посвященной внешним устройствам.

После выборки команды останова ЭВМ прекращает обработку программы. Для выхода из этого состояния требуется либо запрос от внешних устройств, либо перезапуск машины.

Примечание

Интересно, что команда останова активно использовалась для завершения работы программы лишь в первых поколениях ЭВМ. Встретив эту команду, машина прекращала всякую деятельность и ожидала "вмешательства" человека. В современных процессорах команда останова по традиции сохраняется, однако она практически не нужна: компьютер в любой момент должен быть готов продолжить диалог с человеком безо всяких дополнительных действий с его стороны. Иными словами, после завершения работы, допустим, программы текстового редактора, процессор не останавливается, а переходит к выполнению другой программы более высокого уровня — операционной системы. Последняя прекращает свое функционирование лишь перед выключением питания компьютера. Подробнее о вопросах взаимоотношения различных программ мы поговорим в *лекции 7*.

Таков универсальный алгоритм исполнения программы в ЭВМ. Очевидно, что все разнообразие поведения процессора сосредоточено на последнем этапе, когда исполняемые действия определяются считанной командой программы.

Основной алгоритм работы процессора четко и доступно формулируется в учебнике [32]. В качестве более детального описания, снабженного подробными иллюстрациями, можно порекомендовать популярные книги [57, 60].

3.2.2. Проблема начала работы ЭВМ

Если внимательно изучить описанный алгоритм, то можно заметить, что в нем не задано первоначальное значение счетчика адреса команд. Данная неопределенность разрешается следующим образом. При включении питания или при нажатии на кнопку сброса (начальной установки) в счетчик

аппаратно заносится стартовый адрес находящейся в ПЗУ программы инициализации всех устройств и начальной загрузки ЭВМ. Дальнейшее функционирование компьютера определяется программой, которая там записана.

Уместно подчеркнуть, что программный способ управления компьютером делает его необычайно гибким и позволяет на одном и том же оборудовании решать самые разнообразные задачи (включая даже такие, которые в момент изготовления машины еще не были известны!). Роли программного обеспечения в работе вычислительной техники мы посвятим отдельную лекцию.

3.2.3. Организация ветвлений

Основной алгоритм работы ЭВМ позволяет шаг за шагом выполнить хранящуюся в ОЗУ линейную программу. Но этого совершенно недостаточно, поскольку большинство практических задач требуют при обработке данных разветвлений и повторений. Для изменения порядка вычислений в системе команд любой ЭВМ существуют специальные инструкции переходов. Их суть заключается в том, что они в ходе выполнения заносят в счетчик команд необходимый адрес, который задается тем или иным способом в качестве параметра команды перехода. Тем самым мы получаем мощный механизм для произвольного изменения порядка выполнения команд в программе.

Как вам, наверно, известно, переходы бывают *безусловные*, выполняемые всегда, и *условные*, которые совершаются только в случае истинности определенного условия: например, при отрицательности результата.

Примечание

Строго говоря, условными могут быть не только команды переходов, но, например, переписи данных или даже арифметические операции. Однако на практике такое расширение системы команд в ЭВМ встречается не так часто.

В случае если условие не выполняется, инструкция условного перехода игнорируется, и стандартным образом происходит выборка следующей команды программы. Хочется напомнить, что анализ условий осуществляется в арифметико-логическом устройстве *(см. разд. 3.1.2)*.

Добавим к сказанному, что по способу задания адреса, на который необходимо перейти, инструкции делятся на *абсолютные* и *относительные*. В абсолютных переходах адрес задается явно, а в относительных — указывается так называемое смещение, которое прибавляется к текущему содержимому программного счетчика. Разницу между ними легко понять из следующей аналогии. Пусть вы спросили прохожего, где на этой улице ближайшая аптека. На ваш вопрос он может ответить двумя способами: "в доме номер 24" или "через два дома от этого". Описанные варианты и есть не что иное, как абсолютное и относительное указание местоположения объекта.

Важную роль в программном обеспечении играют *переходы с возвратом*, когда процессор запоминает адрес, где произошел переход, и по специальной команде способен возвратиться для продолжения вычислений. Описанный механизм позволяет оформлять часто используемые действия в виде самостоятельных блоков (подпрограмм) и обращаться к ним там, где это требуется. По-моему, это очень напоминает процесс обращения к справочной литературе при чтении сложной книги: нашел нужный термин и продолжил читать.

3.2.4. Оптимизация выполнения команд

В современных компьютерах для ускорения основного цикла выполнения команды используются различные методы.

Конвейеризация. Если внимательно посмотреть на основной алгоритм процессора, то сразу становится очевидным неоптимальная загрузка устройств УУ в ходе его работы. Например, команда извлекается только на первом шаге, а на втором и третьем этапе электронные схемы, ответственные за это, в основном простаивают. Естественно приходит в голову мысль сделать работу устройств УУ более интенсивной. Проще говоря, пока происходит расшифровка и выполнение первой команды, можно извлечь из памяти одну или даже несколько следующих. Такой способ действительно похож на заводской конвейер, когда каждый рабочий выполняет одну простую операцию и передает результаты своего труда другим. Эффект заключается в том, что на всей длине конвейера происходит одновременная параллельная работа над несколькими изделиями сразу.

Очевидно, что конвейер эффективно функционирует только тогда, когда он целиком заполнен. На начальной же стадии первые рабочие заняты, а остальные простаивают. Применительно к процессору этот печальный факт еще более важен в связи с наличием команд переходов, каждая из которых нарушает работу конвейера и требует его "повторного запуска". Некоторая компенсация данного недостатка может быть достигнута за счет применения следующего метода оптимизации — суперскалярности.

Суперскалярность. Суть этого метода, несмотря на некоторую таинственность названия, проста: выражаясь житейским языком, это дублирование устройств. Так, процессор Pentium имеет два конвейера выполнения команд [14], благодаря чему он может выполнять одновременно две инструкции. Кстати, это позволяет существенно смягчить проблему переходов, о которой только что говорилось. В самом деле, встретив команду перехода, процессор на первом конвейере продолжает работы на случай, если переход не произойдет, а второй конвейер запускает с адреса, на который переход может произойти. Разумеется, хотя звучит все это просто и понятно, синхронизация работы двух конвейеров достаточно сложная задача, особенно если

учесть, что последующим командам могут требоваться результаты предыдущих.

Хороший и подробный разбор примеров работы конвейера при одновременном выполнении двух команд приведен в книгах [73] и [68].

Кэширование. Данный прием больше относится к организации памяти, поэтому отложим его рассмотрение до следующей лекции.

3.2.5. Тактовая частота

Любая операция процессора (машинная команда) состоит из отдельных элементарных действий — *тактов*. В зависимости от сложности, команда может быть реализована за разное количество тактов. Например, пересылка информации из одного внутреннего регистра процессора в другой выполняется за несколько тактов, а для перемножения двух чисел их требуется примерно на порядок больше. Существенное удлинение команды происходит, если обрабатываемые данные не находятся внутри процессора и их приходится считывать из памяти.

Для организации последовательного выполнения требуемых тактов в компьютере имеется специальный генератор импульсов, каждый из которых инициирует очередной такт машинной команды (какой именно, определяется устройством процессора и логикой выполняемой операции). Очевидно, что чем чаще следуют импульсы от генератора, тем быстрее будет выполнена операция, состоящая из фиксированного числа тактов. Из сказанного следует, что *тактовая частота* определяется количеством тактовых импульсов в секунду. В настоящее время она чаще всего измеряется в мегагерцах — т. е. миллионах импульсов за 1 сек. Разумеется, эту частоту нельзя установить произвольно высокой, поскольку процессор может просто не успеть выполнить действие очередного такта до прихода следующего импульса. Однако инженеры делают все возможное для увеличения значения этой характеристики, и на данный момент тактовая частота самых современных процессоров уже превышает 1000 МГц, т. е. 1 ГГц (1 гигагерц). Очевидно, что в недалеком будущем "повседневной" единицей тактовой частоты станет именно гигагерц.

Следует четко представлять, что сравнение тактовых частот позволяет надежно определить, какой из двух процессоров более быстродействующий только в том случае, если оба процессора устроены примерно одинаково. Если же попытаться сравнить процессоры, произведенные разными изготовителями и работающие по разным принципам, можно получить абсолютно неправильные выводы. В самом деле, если в одном из процессоров команда выполняется за 2 такта, а в другом — за 3, то при совершенно одинаковой частоте первый будет работать в полтора раза быстрее! Кроме того, не нужно забывать, что производительность современной компьютерной системы

определяется не только быстродействием отдельно взятого процессора, но и скоростями работы остальных узлов компьютера и даже способами организации всей системы в целом: очевидно, что чрезмерно быстрый процессор будет вынужден постоянно простаивать, ожидая, например, медленно работающую память; или другой пример — часто простое увеличение объема ОЗУ дает гораздо больший эффект, чем замена процессора на более быстрый.

3.3. Система команд процессора

3.3.1. Основные группы команд

Несмотря на большое число разновидностей ЭВМ, на самом низком (машинном) уровне системы их команд имеют много общего. Любая ЭВМ обязательно содержит следующие группы команд (взгляните еще раз в табл. 1.3 и 1.4):

1. **Команды передачи данных (перепись)**, копирующие информацию из одного места в другое.
2. **Арифметические операции**, которым фактически обязана своим рождением вычислительная техника. Конечно, доля вычислительных действий в современном компьютере заметно уменьшилась, но они по-прежнему играют в программах важную роль.
3. **Логические операции**, позволяющие компьютеру производить анализ получаемой информации. После выполнения такой команды часто следует условный переход. Простейшими примерами команд рассматриваемой группы могут служить сравнение, а также известные логические операции И, ИЛИ, НЕ (инверсия). Кроме того, к ним часто добавляют анализ отдельных битов кода, их сброс и установка. Подробнее использование логических инструкций процессора будет обсуждаться в *разд. 3.6.4*.
4. **Сдвиги** двоичного кода влево и вправо. Для доказательства важности этой группы команд вспомним еще раз способ двоичного умножения, описанный в *разд. 2.5.5*. В некоторых частных случаях умножение и деление вообще может быть заменено сдвигом (действительно, из повседневного опыта известно, что если дописать или убрать ноль справа, фактически сдвигая десятичную запись числа, то можно увеличить или уменьшить его в 10 раз).
5. **Команды ввода и вывода** информации для обмена с внешними устройствами.
6. **Команды управления**, к которым, прежде всего, следует отнести все виды переходов. Часто сюда же включают операции по управлению процессором.

3.3.2. Процессоры RISC- и CISC-архитектуры

По устройству системы команд процессоры делятся на две большие группы: RISC и CISC. Данные аббревиатуры, которые часто встречаются в компьютерных изданиях, расшифровываются следующим образом: *RISC* (Reduced или Restricted Instruction Set Computer) — компьютер с сокращенным набором команд и *CISC* (Complete или Complex Instruction Set Computer) — компьютер с полным набором команд. Первоначально микропроцессоры имели CISC-архитектуру, для которой характерен набор сложных команд неодинаковой длины с большим количеством методов адресации к памяти. Появившийся позднее RISC-подход предлагал менее сложные команды одинаковой длины с отказом от некоторых сложных методов адресации. В частности, в процессорах с такой организацией обращение к ячейкам памяти производится только двумя специальными командами чтения и записи, а все остальные операции работают с регистрами. Описанные упрощения системы команд позволяют оптимизировать их выполнение и существенно ускорить работу процессора. Что касается "отброшенных" при упрощении возможностей, то они используются относительно редко и вполне могут быть реализованы программным путем.

В *лекции 8* мы подробнее познакомимся с принципами организации RISC-системы команд на примере модели MMIX, предложенной известным американским ученым Д. Кнутом *(см. разд. 8.2.4)*.

Процессоры фирмы Intel относятся к CISC-группе. Однако для того, чтобы наращивать быстродействие своих изделий, Intel широко применяет достижения RISC-архитектуры. В частности, начиная с модели 486, процессоры имеют внутреннее RISC-ядро [14], которое способно эмулировать сложную CISC-систему команд семейства. Рассмотренный в *разд. 3.2.4* конвейер служит еще одним примером.

3.3.3. Структура команд

Любая команда ЭВМ обычно состоит из двух частей — *операционной* и *адресной*. Операционная часть (иначе она еще называется *кодом операции* — КОП) указывает, какое действие необходимо выполнить с информацией. Адресная часть описывает, где используемая информация хранится и куда поместить результат. У некоторых немногочисленных команд управления работой машины адресная часть может отсутствовать, например в команде останова; операционная часть имеется всегда.

Код операции можно представить себе как некоторый условный номер в общем списке системы команд. В основном этот список построен в соответствии с определенными внутренними закономерностями, хотя они и не всегда очевидны.

Рассмотрим теперь подробнее адресную часть.

Первые ЭВМ имели наиболее простую и наглядную трехадресную систему команд. Например: взять числа из адресов памяти A1 и A2, сложить их и сумму поместить в адрес A3.

Трехадресная команда легко расшифровывалась и была удобна в использовании, но с ростом объемов ОЗУ ее длина становилась непомерно большой. Поэтому появились двухадресные машины, длина команд в которых сокращалась за счет исключения адреса записи результата. В таких ЭВМ результат операции оставался в специальном регистре (сумматоре) и был пригоден для использования в последующих вычислениях. В некоторых машинах результат записывался вместо одного из операндов.

Дальнейшее упрощение команды привело к созданию одноадресных машин. Рассмотрим конкретный пример использования такой ЭВМ. Пусть надо сложить числа, хранящиеся в адресах ОЗУ A1 и A2, а сумму поместить в A3. Для решения задачи одноадресной машине потребуется выполнить три команды:

- извлечь содержимое ячейки A1 в сумматор;
- сложить сумматор с числом из A2;
- записать результат из сумматора в A3.

Может показаться, что одноадресной машине всегда нужно втрое больше команд, чем трехадресной. На самом деле это не так. Попробуйте самостоятельно спланировать программу вычисления выражения Y=(X1+X2)*X3/X4 и вы с удивлением обнаружите, что потребуется 3 трехадресных команды и всего 5 одноадресных. Таким образом, одноадресная машина в чем-то даже эффективнее, т. к. она не производит ненужной записи в память промежуточных результатов.

Ради полноты изложения следует сказать о возможности реализации безадресной (нуль-адресной) машины, использующей особый способ организации памяти — стек. Понимание принципов устройства таких машин потребовало бы достаточно подробных разъяснений; в то же время сейчас они применяются довольно редко. Поэтому ограничимся лишь упоминанием того факта, что устроенная похожим образом система команд лежала в основе некоторых программируемых микрокалькуляторов типа Б3-21 и Б3-34 и им подобных.

3.3.4. Пример программы в командах процессора

В заключение, чтобы читатель мог составить хотя бы некоторое представление о том, как выглядит настоящая программа на языке процессора, приведем маленький пример. Перед вами короткая программа для процессора

семейства Intel, которая увеличивает число, находящееся в регистре AX, в 10 раз. Алгоритм не совсем тривиальный, поскольку вместо умножения используются значительно более "быстрые" команды сдвига влево. Каждый такой сдвиг на один разряд эквивалентен умножению на 2, поэтому читатель без труда проследит, как получается требуемый результат.

Программа приводится в табл. 3.2. В первом столбце указаны адреса команд (обратите внимание, что длина команд неодинакова!), а во втором — двоичный код команд процессора (именно такой код на самом деле и исполняет процессор). Следующий (шестнадцатеричный) столбик фактически повторяет содержимое предыдущего, но в более компактном виде. Затем следует общепринятая ассемблерная мнемоника и расшифровывается содержимое операций. Наконец, в последнем столбце даны пояснения к вычислениям.

Таблица 3.2. *Пример программы в командах процессора*

Адрес	Bin-код	Hex-код	Мнемоника	Операция	Результат
100	1101 0001 1110 0000	D1 E0	SHL AX,1	Сдвиг AX влево на один разряд	2 Z
102	1000 1001 1100 0011	89 C3	MOV BX,AX	Скопировать AX в BX	2 Z
104	1100 0001 1110 0000 0000 0010	C1 E0 02	SHL AX,02	Сдвиг AX влево на два разряда	8 Z
107	0000 0001 1101 1000	01 D8	ADD AX,BX	Сложить AX с BX	10 Z

Примечание

Несмотря на то, что приведенная программа по длине явно больше, чем простое умножение на 10, работать она, как не странно, будет заметно быстрее, поскольку умножение — очень "медленная" операция. Не верите? Вот оценка для процессора Intel 80486. По данным, приведенным в [7], команды переписи и сложения типа "регистр-регистр" выполняются очень быстро и требуют всего по одному машинному такту, а сдвиг регистра — три. Несложно подсчитать, что все вычисления по нашей программе будут выполнены в течение 8 тактов работы процессора. В то же время команда умножения целых чисел не может быть выполнена менее, чем за 13 тактов! Для младших моделей семейства, где умножение не было тщательно оптимизировано, разница была еще более существенной. Таким образом, налицо явная экономия времени. Возможности экспериментальной проверки данного утверждения рассматриваются в *разд. 3.6.3*.

3.4. Выводы

Подведем итоги лекции.

- Процессор — это блок ЭВМ, предназначенный для автоматического считывания команд программы, их расшифровки и выполнения. Его главными частями являются АЛУ и УУ. Современная технология позволяет изготовить весь процессор в виде единой микросхемы, которую принято называть микропроцессором.
- Важными функциональными узлами процессора являются регистры. Помимо внутренних регистров АЛУ и УУ процессор содержит регистры общего назначения, которые используются программным обеспечением.
- Основными характеристиками процессора являются разрядность и тактовая частота. Они становятся легко понятными и наглядными, если иметь некоторое представление об устройстве и логике работы компьютера.
- Процессор работает по достаточно простому алгоритму. Он определяет последовательность действий, направленных на извлечение из памяти команд программы и их выполнение. Хотя данный алгоритм в основном одинаков для всех ЭВМ, в современных микропроцессорах используются различные методы для ускорения его выполнения. Программное управление работой процессора позволяет легко настраивать универсальное вычислительное оборудование на решение самых разнообразных задач.
- Каждый процессор имеет определенный набор команд. Базовые операции, которые выполняют процессоры, в основном одинаковы, хотя их конкретные реализации в различных типах процессоров различаются. По организации системы команд и степени ее полноты различают RISC- и CISC-процессоры.
- Команда процессора состоит из кода операции, определяющего необходимое действие по обработке информации, и адресной части, показывающей, где взять исходные данные и куда поместить результат. Адресная часть может быть организована по-разному и содержать до трех адресов.

3.5. Вопросы для осмысления

1. Знаете ли вы марку процессоров в компьютерах, которыми пользуетесь? Какова их тактовая частота и разрядность?
2. Вспомните, есть ли у вас дома устройства, содержащие микропроцессоры?
3. Какие преимущества имеет процессор, собранный внутри единой микросхемы? Есть ли отрицательные черты у миниатюризации? Ответ постарайтесь обосновать.
4. Опишите назначение и устройство АЛУ и УУ.

5. Подумайте, какую логическую операцию нужно осуществить над битами сумматора, чтобы установить факт его равенства или неравенства нулю?
6. Как называется элементарная составляющая машинной команды? От чего, по-вашему, может зависеть скорость выполнения команды?
7. Опишите основные этапы выполнения машинной команды (указание: особое внимание обратите на роль счетчика команд).
8. Что мешает повысить быстродействие микропроцессора за счет увеличения количества конвейеров?
9. Какие основные операции входят в состав системы команд любой ЭВМ? Охарактеризуйте каждую из названных групп.
10. Из каких частей состоит команда ЭВМ? Опишите их назначение.
11. Попробуйте составить программу вычислений по указанной в *разд. 3.3.3* формуле для двух- и трехадресной машины.

3.6. Любопытные эксперименты

3.6.1. Какой процессор находится внутри компьютера?

Рассмотрим вопрос: как узнать марку и характеристики центрального процессора (в литературе часто используется английское сокращение CPU — Central Processor Unit), который находится внутри вашего компьютера. Вопрос этот действительно представляет определенный практический интерес, хотя бы потому, что мало просто открыть компьютер и найти на его главной плате микропроцессор: чтобы добраться до маркировки CPU, требуется еще снять с него систему охлаждения. Так что, как иногда бывает, прямой путь не всегда оказывается самым простым.

Цель эксперимента. Выяснить, как можно узнать о марке процессора и его особенностях.

Начнем с того, что пользователь может прочитать ту информацию о процессоре, которую выводит компьютер при анализе собственной конфигурации в момент включения питания или при перезагрузке. Среди многочисленных текстов, которые появляются на экране в этот момент, есть одна-две строчки, посвященных процессору и его частоте. К несчастью, их трудно прочитать, т. к. они быстро исчезают, поэтому необходимо своевременно нажать клавишу <Pause>.

Другой, более комфортный способ получить информацию о процессоре — обратиться к операционной системе. Если вы работаете в ОС Windows, для этого достаточно щелкнуть правой кнопкой мыши по значку **Мой компьютер** и выбрать пункт **Свойства**. Картина, которую вы увидите, будет примерно такой же, как и изображенная на рис. 3.4.

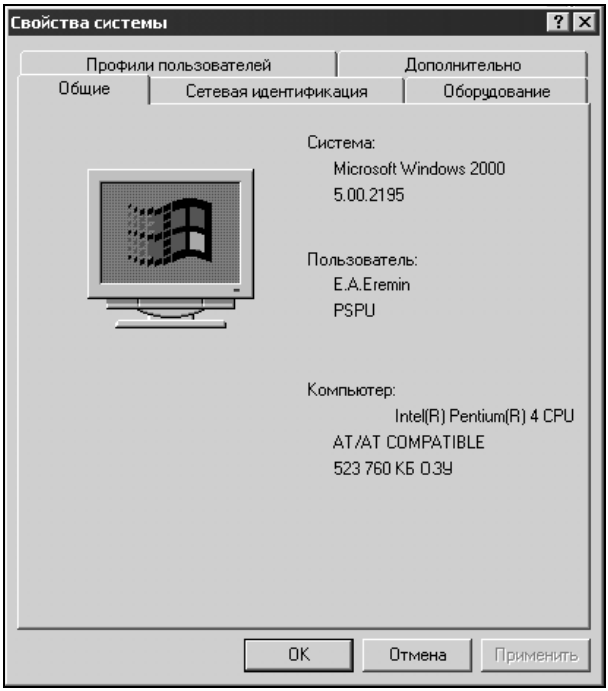


Рис. 3.4. Информация об основных свойствах компьютера

Таким образом, первые и наиболее общие сведения о процессоре мы вывели. Но для наиболее современных типов микропроцессоров можно получить гораздо бóльшую информацию.

Начиная с Pentium, все процессоры для идентификации своей марки оснащены специальной инструкцией `CPUID` [14]. Логика работы этой команды определяется значением, которое заносится в регистр `EAX` непосредственно перед выполнением идентификации. Если установить `EAX = 0`, то процессор в регистрах `EBX`, `EDX` и `ECX` возвращает текстовую строку, определяющую производителя. Так, для процессоров Intel это строка "GenuineIntel", а процессоры, выпущенные фирмой AMD, генерируют строку "AuthenticAMD" ("Genuine" переводится как "подлинный", "настоящий", "неподдельный"; "Authentic" имеет примерно такой же перевод). При `EAX = 1` процессор в этом же регистре возвращает закодированную особым образом более подробную информацию о своем семействе и модели.

Примечание

Подобные данные сразу после включения питания или в момент перезагрузки выдавал в один из регистров уже 386-й процессор, но Pentium способен сообщить ее в любой момент по запросу программы.

Наконец, при вызове инструкции `CPUID` с `EAX = 2` и больше возвращаются дополнительные параметры конфигурации процессора, которые расшифровываются по специальным таблицам. Таким способом можно узнать о наличии математического сопроцессора, поддержке специальных режимов (например, MMX), наличии или отсутствии регистров, свойственных только данной модели, и другую полезную информацию об устройстве CPU.

Повторять все описанные выше манипуляции с регистрами будет для нас, пожалуй, слишком затруднительно. Тем более, что команду `CPUID` распознает только самое современное программное обеспечение: даже программа Debug, входящая в состав Windows 2000, "не понимает" мнемонику данной инструкции. Поэтому выберем более простой вариант. С официального сайта компании Intel загрузим специализированную программу-утилиту для получения информации о процессоре [И-24] и запустим ее. Получится картина, которая, разумеется, зависит от установленного в машине процессора (рис. 3.5).

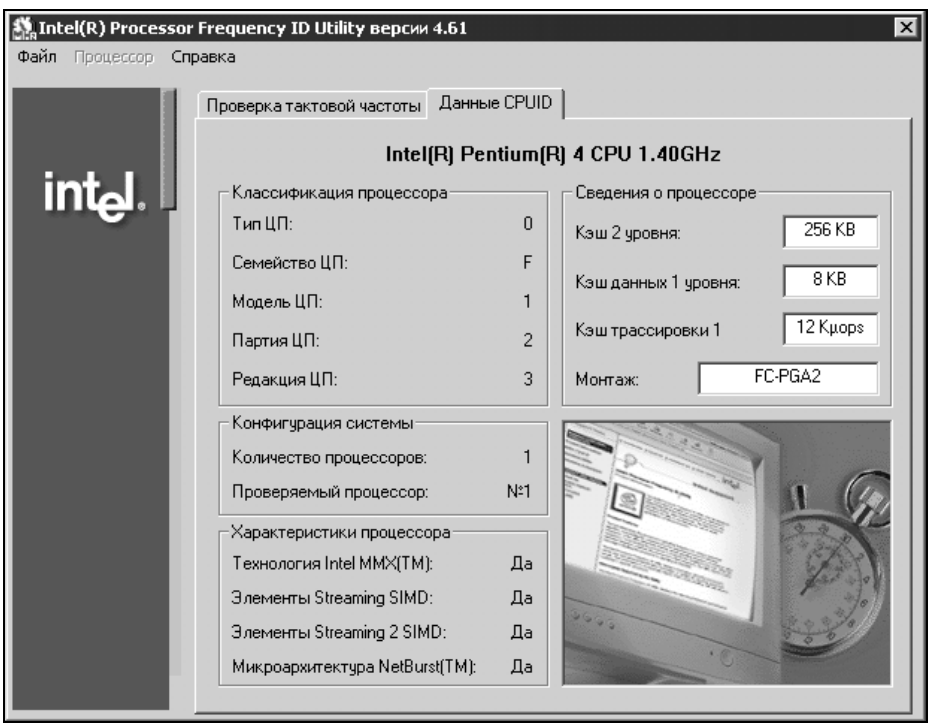


Рис. 3.5. Результаты идентификации процессора с помощью утилиты фирмы Intel

Пояснение на русском языке по каждому отображаемому параметру можно получить в разделе **Справка**. Например, семейство процессора **F** означает Pentium 4, а поля **Модель ЦП**, **Партия ЦП** и **Редакция ЦП** содержат дан-

ные о версии разработки или промышленной партии микропроцессоров; они могут потребоваться при обращении в фирму за технической поддержкой.

Отметим, что имеется большое количество программ, аналогичных рассмотренной выше, которые определяют конфигурацию компьютера, например, [И-22].

Примечание

Наиболее внимательные читатели, наверно, спросят: а сможет ли программное обеспечение распознать более ранние модели? Да, если использует некоторые тонкие особенности их работы. Подробности того, как это можно сделать, также описаны в книге [14].

Интересно, что команда `CPUID` не способна определить частоту процессора. Тем не менее, использованная нами утилита не только отображает "паспортную" тактовую частоту, но и измеряет фактическую частоту для моделей Pentium III и выше.

Вывод. Таким образом, получить информацию о марке и внутреннем устройстве современного процессора можно программным путем. Это позволяет программному обеспечению учитывать особенности установленного в компьютере процессора и корректно оптимизировать свою работу там, где для этого имеются технические возможности.

3.6.2.* Измерение быстродействия процессора

Компьютер выполняет те действия, которые удалось запрограммировать, во много раз быстрее человека. Во сколько раз? Как быстро в количественном отношении работает компьютер? На подобные вопросы трудно ответить однозначно. После слов "процессор выполняет за 1 секунду..." в литературе приводится самое разнообразное количество операций. Причин тому множество: компьютеры сильно отличаются по характеристикам, на разные операции требуется неодинаковое время, благодаря современному многозадачному режиму очень трудно сказать, какая часть времени непосредственно ушла на вашу задачу, и много других.

Попробуем все же оценить, насколько быстро ваш компьютер выполняет ту или иную операцию.

Цель эксперимента. Научиться оценивать количество операций, которое ваш компьютер выполняет за 1 сек.

На пути аккуратного измерения быстродействия компьютера имеется целый ряд довольно значительных трудностей. Сформулируем их ради того, чтобы знать, с чем придется иметь дело при решении поставленной задачи.

- Одна операция компьютера выполняется за незначительную долю секунды: следует ожидать, что современный процессор с частотой 1 ГГц вы-

полняет простейшую команду за время порядка 10^{-9} сек. (величина, обратная частоте).

- Диапазон тактовых частот окружающих нас с вами компьютеров достаточно широк, поскольку рядом с новейшими моделями подчас на соседнем столе стоит компьютер, выпущенный 4—5 лет назад.
- Время выполнения различных команд процессора сильно различается.
- Помимо нашей задачи компьютер параллельно выполняет еще несколько, например, очень существенно влияет функционирование операционной системы.
- Внутренне устройство современного процессора довольно сложное. В частности, для ускорения его работы используются специальные методы (вы читали о некоторых из них в *разд. 3.2.4*), благодаря которым команды программы, следующие друг за другом, могут выполняться параллельно.

Вот далеко не полный перечень трудностей, которые возникают при проведении данного эксперимента. Существенно уменьшить необходимость их учета можно, если не ставить себе целью точное измерение времени выполнения операции, а ограничиться ее оценкой, допустим, по порядку величины. Именно так мы и поставим себе задачу.

Вывод расчетной формулы

Пусть у нас имеется две незначительно различающихся циклических программы `P1` и `P2`, причем цикл в каждой из которых повторяется `N` раз. Предположим также, что программы различаются внутри цикла только одной машинной командой. Тогда, измерив время работы обеих программ `t1` и `t2`, можно приближенно оценить время выполнения данной машинной команды по формуле:

$$T = (t_1 - t_2) / N$$

Если интересоваться количеством выполненных за 1 сек. операций `K`, то надо взять обратную величину:

$$K = N / (t_1 - t_2)$$

Учтем дополнительно, что время по внутреннему таймеру компьютера обычно выдается с точностью до 0,01 сек., а величину `K` удобнее с практической точки зрения измерять в миллионах операций в секунду.

Примечание

Интересно, что традиционная частота таймера равнялась 1/18 сек., т. е. больше 0,01 сек. Для вычисления сотых долей секунды MS-DOS использовал специальный алгоритм. По утверждению Питера Нортона, алгоритм этот достаточно хорош (стр. 280 [61]).

Добавив в знаменатель и в числитель соответствующие множители, после сокращения получим:

$$K = N * 10^{-4} / (t_1 - t_2)$$

в которой K уже выражено в миллионах операций в секунду. Наконец, можно еще несколько упростить расчетную формулу, если положить

$$N = 10^p * 10^4$$

Примечание

В дальнейшем оказалось, что такое представление величины N, уменьшающее используемые числа на 4 порядка, удобно еще и для того, чтобы на быстродействующих компьютерах необходимое количество повторений цикла "поместилось" в переменную типа longint (это тип, позволяющий в Турбо Паскале хранить максимально большие целые числа).

Итак, сомножитель 10^4 сокращается и формула приобретает окончательный вид:

$$K = 10^p / (t_1 - t_2)$$

Значение p, вообще говоря, произвольно, но в расчетах не должно превышать 9, поскольку 10^{10} уже выходит за допустимый для longint диапазон чисел. Этот параметр задает количество повторений цикла и тем самым определяет время проведения теста, поэтому естественно желание взять его поменьше. С другой стороны, слишком маленькие значения p приводят к большой погрешности в измерении коротких интервалов. О процедуре выбора "разумного" значения p будет подробнее рассказано при описании программы.

Программы для проведения эксперимента

Остается научиться разрабатывать описанные выше программы P1 и P2. Листинг 3.1 относится к первой из них. Ее назначение состоит в том, чтобы измерить время исполнения программы-оболочки, к которой позднее будет добавлена исследуемая машинная команда. Поскольку тестировать придется разные по скорости машины, программа P1 "подбирает минимально возможное разумное значение" константы p: для надежности измерения временной интервал должен быть больше 1 000 (10 сек.). Время t1 и вычисленная по полученному p константа для организации циклов сохраняются в файл с именем test.txt.

Листинг 3.1. Программа измерения времени исполнения команд процессора

```
PROGRAM instruction_time_P1;
USES dos;
VAR h,m,s,s100, h2,m2,s2,s1002:WORD; p,j:INTEGER;
    n4,n, w1,w2,w3,w4,d:LONGINT; f:TEXT;
```

```
BEGIN n4:=1; p:=0;
REPEAT  n4:=10*n4; p:=p+1; WRITELN(p,' ',n4); {n4=10^p}
      {N=10000*n4 — полное количество циклов тестирования}
      GetTime(h,m,s,s100); {***** begin test *****}
      FOR n:=1 TO n4 DO FOR j:=1 TO 10000 DO
          BEGIN asm   mov ax,1234h
                      mov bx,4321h
                end;
          END;
      GetTime(h2,m2,s2,s1002); {**** end test ****}
      WRITELN('start:',h:3,m:3,s:3,s100:3);
      WRITELN('stop: ',h2:3,m2:3,s2:3,s1002:3);
      w4:=m2; w2:=360000*h2+6000*w4+100*s2+s1002;
      w3:=m; w1:=360000*h+6000*w3+100*s+s100; d:=w2-w1;
      WRITELN('d=',d);
UNTIL (d>1000) or (p=9); {достаточное или максимальное время}
      ASSIGN(f,'test.txt'); REWRITE(f); {записать в файл}
      WRITELN(f,d); WRITELN(f,n4); WRITELN(f,p); CLOSE(f);
END.
```

Внешний цикл `REPEAT` организует перебор значений `p`, начиная с 1. Для каждого из них проделываются следующие описанные ниже действия.

С помощью стандартной процедуры Паскаля `GetTime` фиксируется время начала теста — в переменные `h`, `m`, `s` и `s100` заносятся значения часов, минут, секунд и сотых долей секунды в момент старта. Далее два вложенных цикла `FOR` обеспечивают повторение 10^4 * `n4` раз некоторого ассемблерного фрагмента, заключенного между служебными словами `ASM` и ближайшим к нему `END`. Пока этот фрагмент просто задает значения (буква `h` в конце означает, что они шестнадцатеричные) регистров `AX` и `BX` микропроцессора. Позднее в программе `P2` к нему будет дописана инструкция, время которой мы хотим измерить. Момент завершения всех циклов тестирования снова фиксируется функцией `GetTime` в переменные с названиями, аналогичными перечисленным выше, только с добавленной в конце цифрой 2.

После этого для контроля выводятся оба показания таймера, и вычисляется разность между ними. Сделать это непосредственно достаточно затруднительно, т. к. каждое показание времени представляет собой набор из четырех чисел. Для упрощения задачи придется преобразовать их к единому целому числу, представляющему собой сотые доли секунды. Сделать это можно по длинной, но вполне тривиальной формуле:

```
w1 = 360000 * h + 6000 * m + 100 * s + s100
```

Числовые коэффициенты в этой формуле переводят все величины в сотые доли секунды. Например, 1 час = 3 600 сек. и еще два нуля добавляются для

перевода в сотые доли. Последнее слагаемое, естественно, не нуждается в преобразовании и не имеет никаких множителей.

Полученное значение `w1` будет достаточно большим, поэтому оно описано в программе как 4-байтовое целое `longint` (как видно из описаний программы, все переменные, входящие в правую часть рассматриваемого выражения, имеют беззнаковый двухбайтовый тип `word`, что требуется для корректного обращения к процедуре `GetTime`).

Уточнение (очень важное, хотя и длинное)

Жизнь всегда сложнее любой самой продуманной книги. Как тщательно автор подыскивал убедительные примеры для вводной лекции, которые бы продемонстрировали читателю необходимость знания принципов работы компьютера *(см. разд. 1.1)*! А тут пример возник в ходе решения задачи сам собой — вот он прямо перед нами! Где? Да вот же, в формуле для `w1`. На первый взгляд ничего особенного в ней нет. Пожалуй, только то, что слева стоит переменная типа `longint`, а справа — исходные данные типа `word`. Но компилятор "не возражает " против этого — он "знает", как выполнить это преобразование. И, тем не менее, результат будет неправильным! Причем только в одном из слагаемых. Думаете, в первом, самом большом? Нет, во втором!!! Чтобы это объяснить, надо хорошо понимать, как машина определяет тип промежуточных результатов: если оба операнда одинаковы, то и результат будет иметь тот же самый тип; в противном случае, будет выбран более общий — например, при перемножении `word` и `longint` в качестве результата будет взят последний. Посмотрим с этой точки зрения на слагаемые в формуле для `w1`. В первом из них коэффициент явно выходит за диапазон типа `word` (превышает 65 635) и число автоматически будет считаться "большим", т. е. `longint`. Произведение также будет иметь этот тип, и все получится правильно. Что же касается всех остальных слагаемых, то они имеют тип `word`, а значит, и произведения будут соответствующими. Видите, где ошибка? Слагаемое, содержащее минуты, будет вычисляться правильно только для малых `m`, точнее, пока `m` < 65 536/6000, что составляет чуть меньше 11 минут. После этого произведение превысит 65 535 и машина примет в качестве ответа ту часть, которая "перешла" через критическое значение: например, вместо 65 538 останется всего 2. Но и это еще не все! Поскольку мы печатаем разность `d = w2 – w1`, то она чаще всего будет правильной, несмотря на описанные странности в вычислениях. В самом деле, и в уменьшаемом, и в вычитаемом будет "потеряна" одна и та же величина, а значит, разность не изменится — перед нами тот редкий случай, когда две ошибки не усиливаются, а компенсируются. Но, к сожалению, будут моменты, когда ошибка все же проявится: если `w2` уже перешло границу и подсчитано неправильно, а `w1` пока еще правильное. Тогда, как нетрудно сообразить, `d` выйдет отрицательным, что сложно не заметить. Причем такие странности будут происходить 5 раз каждый час (вблизи 11, 22, ..., 55 минут каждого часа). Именно благодаря этому проявлению ошибку и удалось заметить. Так что видите, уважаемые читатели, насколько важно понимать логику работы компьютера!

Вернемся к нашей задаче. Если вы прочитали приведенное выше уточнение внимательно, то назначение промежуточных переменных `w3` и `w4` становится понятным: их добавление в формулу заставит компилятор установить пра-

вильный тип второго произведения в ней по этой дополнительной переменной. Вычисленное с такими предосторожностями значение разности d проверяется на достаточность (см. условие после слова UNTIL) и, если условие выполняется, цикл завершается. Он также должен принудительно завершиться и при p = 9, чтобы, как указывалось ранее, не превышать допустимое для longint значение. К счастью, для проверенных процессоров реально полученные значения не превышали 6, так что это пока скорее предосторожность, чем необходимость.

Программа P1 завершается записью подобранных данных в файл test.txt.

Перейдем теперь к программе P2, которая, как следует из общей идеи, должна быть очень похожа на P1. Посмотрев на листинг 3.2, мы можем убедиться в этом.

Листинг 3.2. Модифицированная программа измерения времени исполнения команд процессора

```
PROGRAM instruction_time_P2_add;
USES dos;
VAR h,m,s,s100, h2,m2,s2,s1002:WORD; t1,j:INTEGER;
    n,n4, w1,w2,w3,w4,d:LONGINT; f:TEXT;
BEGIN ASSIGN(f,'test.txt'); RESET(f); {читать из файла}
      READLN(f,t1); READLN(f,n4); CLOSE(f);
      WRITELN('t1=',t1,' N*10^(-4)=',n4);
      {10000*n4 — количество повторений цикла}
      GetTime(h,m,s,s100); {***** begin test *****}
      FOR n:=1 TO n4 DO FOR J:=1 TO 10000 DO
          BEGIN asm   mov ax,1234h
                      mov bx,4321h
                      add ax,bx    {#}
                end;
          END;
      GetTime(h2,m2,s2,s1002); {**** end test ****}
      WRITELN('start:',h:3,m:3,s:3,s100:3);
      WRITELN('stop: ',h2:3,m2:3,s2:3,s1002:3);
      w4:=m2; w2:=360000*h2+6000*w4+100*s2+s1002;
      w3:=m; w1:=360000*h+6000*w3+100*s+s100; d:=w2-w1;
      WRITELN('d=',d);
      WRITELN((n4/(d-t1)):9:3, ' mln op/sec')
END.
```

Программа считывает из приготовленного ранее файла значения констант t1 и n4, а затем делает все то же самое, что и предыдущая программа. Единственное отличие заключается в добавленной команде, которая в комментарии помечена символом # (в листинге 3.2 это команда сложения add ax,bx).

Вместо нее, разумеется, можно поставить и другую команду, например умножения: `mul bx`, что и сделано в приведенном на диске дополнительном файле, который называется P2MUL.

Если читателю показалось сложным приведенное выше детальное объяснение подготовки эксперимента, он может воспользоваться прилагаемым к книге CD-диском, где приведены готовые исполняемые файлы.

Обсуждение результатов эксперимента

Наконец-то все готово и мы можем перейти непосредственно к выполнению эксперимента. Запустим программу `P1` (согласно принятой системе имен, на CD-диске она называется по номеру листинга LIST_3_1), которая подготовит в файле test.txt необходимые для проведения тестирования константы. После этого выполним несколько раз программу `P2` (LIST_3_2) и усредним полученные значения. Затем заменим приведенную в листинге 3.2 команду сложения на умножение и повторим тестирование.

Автор проверил описываемую методику определения времени выполнения одной инструкции процессора на большой группе компьютеров. Результаты экспериментов сведены в табл. 3.3.

Таблица 3.3. *Время выполнения инструкций различными процессорами*

Процессор	Частота (МГц)	ОС	p	t1	add ax,bx	mul bx
					(млн. операций в сек.)	
Intel 80286	7	DOS	3	5251	3	0,4
Intel 80486SX	25	DOS	4	8816	12	0,5
Intel 80486DX2	66	DOS	4	1648	22	2
Pentium	120	DOS	5	5844	120	11
		Windows 95	5	6135	100	10
Pentium MMX	166	DOS	5	3620	170	15
		Windows 95	5	3740	130	15
Pentium MMX	200	DOS	5	3015	200	18
		Windows 98	5	3054	190	18
Pentium MMX	233	DOS	5	2565	230	20
		Windows 98	5	2850	215	19

Проанализируем полученные результаты. В табл. 3.3 представлены данные об экспериментах над различными процессорами — от 80286 до Pentium. Сразу подчеркнем, что на процессорах более старших моделей стабильных

хорошо воспроизводящихся результатов получить не удалось. По-видимому, сказывается заметное усложнение их архитектуры, особенно то, что инструкции могут исполняться не совсем в том порядке (out of order), в каком они следуют в программе [14]. Из-за этого добавление к программе одной ассемблерной команды совсем не обязательно приводит к увеличению времени ее исполнения. Наоборот, если операция простая, то программа `P2` на компьютерах с Celeron, Pentium II и выше часто начинала выполняться быстрее(!), чем `P1`. Особенно плохо было то, что измеряемое время зависело от операторов, вне теста: например, стоило убрать вывод на экран показаний таймера, и время выполнения программы изменялось, причем не всегда в меньшую сторону.

Существенное влияние на результаты оказывала среда, в которой исполнялись программы. Эксперименты проводились с различными версиями DOS (включая ее эмуляцию в Windows и весьма экзотическую версию Caldera DR-DOS [45]) и Windows вплоть до версии 2000 года. Как и следовало ожидать, сложная операционная система отбирает у задачи часть ресурсов и уменьшает эффективную скорость решения задачи. Особенно сильное замедление наблюдалось в Windows 2000. Напротив, во всех версиях DOS результаты оказались практически одинаковыми и очень стабильными, причем, как видно из табл. 3.2, количество операций получилось выше.

Столбцы `p` и `t1` приводятся для возможности повторения экспериментов и показывают лишь очевидный рост производительности с увеличением тактовой частоты. Основные результаты представлены в двух последних столбцах, где собраны результаты измерения времени выполнения двух характерных операций — сложения и умножения. Напомним читателю, что частота в мегагерцах фактически означает количество миллионов тактов за одну секунду (1 МГц = 1 000 000 герц), поэтому имеет смысл сопоставлять приводимые результаты с соответствующими значениями тактовой частоты. Так для уже практически вышедшего из употребления процессора 80286 время выполнения сложения оказалось эквивалентным двум тактам работы процессора, а для умножения их около 20. Из литературы [18] действительно следует, что сложение регистр-регистр выполняется за 2 такта, а умножение для 16-разрядного регистра — за 21 такт. С процессором 80486 такого соответствия с приводимыми в технической литературе данными не получается. Например, по данным книги [7], сложение должно выполняться за один такт (из таблицы отчетливо получаются значения 2 и 3). В неоднократно цитированной в данном примере книге [14] также утверждается, что в рассматриваемом процессоре "применено RISC-ядро, позволяющее наиболее часто встречающиеся инструкции выполнять за 1 такт". Автор затрудняется объяснить полученное несоответствие, тем более, что эксперименты с командой `mov ax,bx` на машине с частотой 67 МГц дали результат 67 млн. операций в секунду, что соответствует одному машинному такту на

операцию. Давайте все же вспомним, что мы собирались не измерять, а оценивать время выполнения инструкций — для оценки результат вполне годится.

Зато для процессоров типа Pentium эксперимент дал на удивление хорошие результаты. Как видно из таблицы, значения в столбцах с тактовой частотой и количеством операций сложения практически совпадают.

Выводы. Из полученных результатов следует целый ряд интересных и важных выводов.

- Производительности процессора при прочих равных условиях действительно пропорциональна тактовая частоте.
- Архитектура процессора также существенно влияет на его производительность. Например, описанная методика прекрасно сработала для процессоров Pentium, зато оказалась абсолютно непригодной для процессоров Celeron.
- На скорость выполнения задачи заметно влияет программная среда. Операционная система, созданная для распределения ресурсов, сама также их уменьшает. Сказанное в полной мере применимо и к быстродействию процессора.
- Разные операции требуют разного времени для выполнения. Традиционно "длинной" инструкцией является умножение, на которое, несмотря на все усилия инженеров в этом направлении, по-прежнему уходит времени примерно на порядок больше, чем на сложение.
- Можно считать, что на современных процессорах простые инструкции (пересылки, сложение регистров и т. п.) выполняются примерно за один такт. Таким образом, тактовая частота вполне может служить приблизительной оценкой количества таких операций, выполняемых за 1 сек.

3.6.3. Экспериментальное сравнение эффективности программ

Используя методику измерения промежутков времени, подробно развитую в предыдущем эксперименте, можно сравнивать между собой эффективность небольших алгоритмов, составленных в командах процессора. Например, в данной лекции в *разд. 3.3.4* приведена программа из четырех машинных инструкций, умножающая содержимое регистра `AX` на 10. На основании справочных данных утверждается, что она будет выполняться быстрее, чем стандартное умножение.

Цель эксперимента. Проверить приведенный в лекции тезис и убедиться в возможности применения предложенной методики для сравнения эффективности программ.

Слегка модифицируя программу, приведенную в листинге 3.1, легко получить новую, которая выполняет тестирование двух разных фрагментов кода и сравнивает время их работы. Читатели могут попробовать проделать этот эксперимент самостоятельно. Авторский вариант программы находится на прилагаемом к книге CD-диске в файлах cmp_mul.pas и cmp_mul.exe.

Экспериментальное сравнение эффективности двух алгоритмов умножения на 10 показало, что, в соответствии с утверждением лекции, стандартное умножение заметно дольше. Несколько пробных экспериментов на разных компьютерах подтвердили уменьшение времени вычислений в 1,2—1,9 раза.

Аналогичная проверка может быть произведена и для похожей задачи, рассмотренной в *разд. 2.5.5*.

Вывод. Эксперименты подтверждают правильность положения, высказанного и теоретически обоснованного в лекции. Предложенная методика действительно дает возможность сравнивать время выполнения двух фрагментов в машинных кодах.

3.6.4. Как используются логические инструкции процессора

Вы, конечно, обратили внимание на то, что в систему команд любой ЭВМ входят логические операции. Некоторые современные пользователи компьютера считают, что данная группа инструкций была введена исключительно для работы со сложными логическими выражениями. При этом они живо вспоминают составление запросов к базе данных и другие подобные применения логических выражений, соединенные операциями И, ИЛИ, НЕ. Тем не менее обсуждаемая группа операций появились задолго до работы с логическими данными и с совершенно другими целями. Давайте на практике познакомимся с ними.

Цель эксперимента. Выяснить, где и как можно использовать логические операции.

Обсуждение начнем с того, что обратим внимание на важный факт: в отличие от арифметических операций, логические команды являются ***поразрядными***. Например, при сложении возможен перенос в старший разряд, а при логических операциях все разряды рассматриваются изолированно друг от друга. Разумеется, действия над всеми разрядами выполняются параллельно и одновременно.

Над каждым разрядом производятся действия по следующим широко известным правилам (табл. 3.4).

Посмотрим, как этим можно воспользоваться на практике. Для удобства и ясности изложения договоримся называть первый операнд `X` данными, а операнд `Y` — "маской", с помощью которой мы будем воздействовать на данные.

Таблица 3.4. *Правила выполнения логических операций*

X (данные)	Y (маска)	X AND Y	X OR Y	X XOR Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Начнем с логической операции И. Внимательно изучив содержимое табл. 3.4, можно заметить, что если разряд маски равен нулю, то независимо от своего первоначального содержимого разряд данных в результате операции сбрасывается. Если же маска, напротив, единичная, то соответствующий разряд данных не изменяется. Таким образом, логическая операция И способна выделить (сохранить) только те разряды, для которых в маске указаны единички, сбрасывая в ноль все остальные.

Аналогичным образом рассмотрим операцию ИЛИ. Как вы, конечно, догадались, для нее, наоборот, сохраняются без изменений те разряды, для которых в маске указан ноль. В противном случае результат всегда будет единичным. Итак, с помощью логической операции ИЛИ можно дополнительно установить единицу в тех разрядах, где в маске установлены единицы. Таким образом, операция И сбрасывает требуемые биты, а ИЛИ, наоборот, устанавливает их. Конечно, можно комбинировать обе операции, что позволит произвольным образом формировать любую интересующую нас часть двоичного кода, не меняя остальные разряды.

Наконец, операция `XOR` ("eXclusive OR" — исключающее ИЛИ) незаменима при сравнении бинарных кодов. Из табл. 3.4 следует, что при совпадении разряда данных с разрядом маски он сбрасывается, в противном случае устанавливается в единицу. Заметим, что если все разряды операндов одинаковы, то результат тождественно нулевой.

Для проверки всех этих рассуждений служит следующая программа (листинг 3.3).

Листинг 3.3. Программа для изучения логических инструкций процессора

```
PROGRAM logic;
CONST num=$a0f0; mask=$50ff;   VAR   i:INTEGER;
PROCEDURE binPrint(b:WORD); {двоичный вывод}
VAR   m:WORD;
BEGIN m:=$8000; {bin = 10...00}
      FOR i:=1 TO 16 DO
```

```
          BEGIN IF (b AND m) = 0
                   THEN WRITE('0')
                   ELSE WRITE('1');
                IF (i MOD 4)=0 THEN WRITE(' ');
                m:=m SHR 1
          END;  WRITELN
END;
BEGIN binPrint(num); binPrint(mask);
      WRITELN('логическая операция OR: ');
      binPrint(num OR mask);
      WRITELN('логическая операция AND: ');
      binPrint(num AND mask);
      WRITELN('логическая операция XOR: ');
      binPrint(num XOR mask);
END.
```

Процедура `binPrint` служит для вывода двоичного кода числа `b` на дисплей. В ней, кстати, тоже используется маска `m` для поочередного выделения каждого бита. После выделения текущего бита маска сдвигается вправо (операция `SHR`) и все циклически повторяется. Отметим, что условие `(i MOD 4) = 0` позволяет поставить пробел после каждой "четверки" двоичных цифр.

Основная программа тривиальна: с помощью процедуры `binPrint` выводятся на экран необходимые данные, сопровождаемые текстовыми комментариями.

Примечание

Исходные данные для программы начинаются со значка "$", что означает шестнадцатеричную систему счисления.

Результат работы для приведенных в листинге данных будет следующим:

1010 0000 1111 0000
0101 0000 1111 1111

логическая операция `OR`:

1111 0000 1111 1111

логическая операция `AND`:

0000 0000 1111 0000

логическая операция `XOR`:

1111 0000 0000 1111

Изменяя исходные данные, можно проверить следующие свойства логических операций:

- результат операции И при нулевой маске равен нулю, а при маске, состоящей из одних единиц, данные при выполнении операции не меняются;

- результат операции ИЛИ при маске из всех единиц будет полностью единичным, а нулевая маска не дает никакого эффекта;
- операция И с числом 1 выделяет содержимое младшего разряда;
- операция исключающее ИЛИ является "обратимой" — ее повторное применение восстанавливает первоначальное число;
- исключающее ИЛИ между одинаковыми числами дает нулевой результат;
- исключающее ИЛИ с маской из всех единиц заменяет все разряды на противоположные (имитирует операцию НЕ).

Какие практические задачи можно решить с помощью логических операций? Приведем пару примеров.

Рассмотрим важный для практики процесс преобразования регистра символов (перевод из заглавных букв в строчные и обратно). Поскольку регистр буквы определяется состоянием определенного бита, на него легко воздействовать с помощью логических операций.

Ниже приведена программа, иллюстрирующая смену регистра латинских букв (листинг 3.4). В силу некоторых особенностей кодировки для русских букв задача решается сложнее, и мы не будем здесь ее анализировать.

Листинг 3.4. Программа, иллюстрирующая смену регистра латинских букв

```
PROGRAM change_case;
VAR s:STRING; i:INTEGER;
BEGIN WRITELN(' Input text:');READLN(s);
      FOR i:=1 TO length(s) DO
          IF (s[i]>='a') AND (s[i]<='z')
             THEN s[i]:=chr(ord(s[i]) AND $df);
      WRITELN(s);
      FOR i:=1 TO length(s) DO
          IF (s[i]>='A') AND (s[i]<='Z')
             THEN s[i]:=chr(ord(s[i]) OR $20);
      WRITELN(s);
END.
```

Как видно из листинга программы, если символ нуждается в преобразовании (является строчной латинской буквой в первой части и заглавной латинской во второй), над ним проделываются следующие манипуляции: преобразуют в число с помощью функции `ord()` и после необходимой логической операции снова возвращают в символьную форму функцией `chr()`.

Приведенная программа замечательна еще и тем, что в ней логические операции используются и в другом своем качестве — как средство организации составных условий в операторе `IF`. Автор настоятельно рекомендует читате-

лю тщательно проанализировать оба применения логических операций и запомнить разницу между ними.

Приведем наконец пример результата работы программы. После ввода предложения "This is my IBM PC computer!" на экране появятся результаты преобразования:

THIS IS MY IBM PC COMPUTER!
this is my ibm pc computer!

Подобные манипуляции с битами используются гораздо чаще, чем может показаться на первый взгляд. Назовем, в частности, преобразование цифрового символа в число и наоборот (код цифры "0" — 0011 0000, "1" — 0011 0001 и т. д.), а также поддержку битовых структур данных, например, множеств.

В заключение рассмотрим еще один пример. Учитывая обратимость операции XOR, ее можно использовать для шифрования. На листинге 3.5 приведен простейший вариант решения этой задачи: операция XOR между кодом числа и его номером в строке.

Листинг 3.5. Программа шифрования текста с использованием операции XOR

```
PROGRAM shifr;
VAR s:STRING; i:INTEGER;
BEGIN WRITE  ('Введите  свой  текст: ');READLN(s);
      FOR i:=1 TO length(s) DO s[i]:=chr(ord(s[i]) XOR i);
      WRITELN('зашифрованный  текст: ',s);
      FOR i:=1 TO length(s) DO s[i]:=chr(ord(s[i]) XOR i);
      WRITELN('расшифрованный текст: ',s)
END.
```

Программа предельно проста и фактически является упрощенным вариантом предыдущей. Поэтому автор предоставляет читателям возможность разобраться в листинге 3.5 самостоятельно.

Результат шифрования с помощью такой короткой программы выглядит достаточно впечатляюще (рис. 3.6)

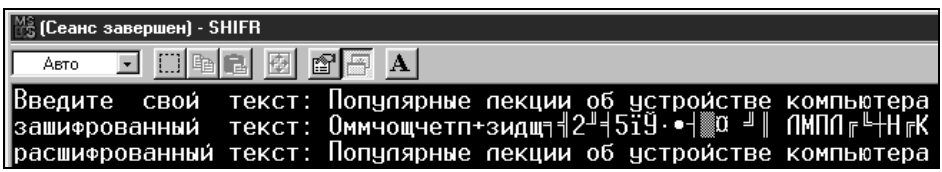
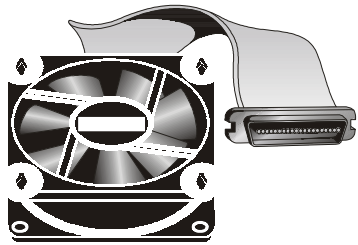


Рис. 3.6. Результат работы программы шифрования

Обратите особое внимание, что одни и те же символы в различных частях текста преобразуются в разные коды.

Вывод. Логические операции могут широко применяться для формирования двоичного кода и, следовательно, для обработки информации. Так операция И позволяет выделить необходимые биты кода и сбросить остальные, а ИЛИ — добавить единицы в требуемую его часть. Много интересных применений имеет также операция исключающее ИЛИ.

ЛЕКЦИЯ 4



Память ЭВМ: ОЗУ

Итак, мы выяснили, что вся работа процессора ЭВМ представляет собой непрерывное выполнение программы. Но в самом процессоре не предусмотрено место для ее хранения целиком, а выбирается лишь та часть, которая в данный момент исполняется. Отсюда следует, что для хранения программ обработки данных к процессору обязательно требуется добавить еще один важный функциональный узел — ***память***.

Память бывает нескольких видов, поэтому в данной лекции рассматриваются в основном вопросы, относящиеся к ***оперативной*** памяти. Не менее интересная с практической точки зрения ***внешняя*** память будет темой следующей лекции.

4.1. Назначение и виды памяти

Память — это одно из основных устройств ЭВМ, которое "используется... для записи, хранения и выдачи по запросу информации, необходимой для решения задачи на ЭВМ" [35]. Еще раз подчеркнем, что в памяти хранятся не только данные решаемых задач, но и программы их обработки.

Существует множество видов памяти, которые различаются по устройству, организации, функциональному назначению и т. д. В нашей лекции мы попробуем охватить наиболее существенные из них. Начнем с самого общего деления — на внутреннюю и внешнюю память.

4.1.1. Внутренняя и внешняя память

Термин этот имеет историческое происхождение и, к сожалению, в настоящее время во многом утратил свою изначальную наглядность. Первые ЭВМ, как мы знаем, представляли собой огромные шкафы, заполненные всевозможными электронными блоками. Та память, которая находилась внутри

центрального процессорного шкафа (или плотно к нему примыкала), получила естественное название *внутренней*. А память, сконструированная в виде отдельных устройств для хранения информации (например, накопители на магнитных лентах и барабанах), не менее естественно стала называться *внешней*. По мере развития технологий производства размеры всех электронных устройств уменьшились настолько, что большинство из них удалось разместить внутри единого корпуса, и прежняя терминология утратила свой первоначальный смысл. Тем не менее указанная классификация памяти сохранилась: в [15] предложена следующая современная трактовка терминов для компьютера IBM PC:

"*Внутренняя память* — электронная (полупроводниковая) память, устанавливая на системной плате или на платах расширения.

Внешняя память — память, реализованная в виде устройств с различными принципами хранения информации и обычно с подвижными носителями. В настоящее время сюда входят устройства магнитной (дисковой и ленточной) памяти, оптической и магнитооптической памяти. Устройства внешней памяти могут размещаться как в системном блоке компьютера, так и в отдельных корпусах".

Несколько забегая вперед, отметим дополнительно, что доступ процессора к внутренней и внешней памяти реализуется по-разному.

В данной лекции мы ограничимся рассмотрением внутренней памяти, а изучение внешней отложим до следующей лекции.

4.1.2. ОЗУ, ПЗУ, ППЗУ и некоторые другие виды памяти

По способам изменения содержимого внутренняя память имеет несколько различных типов. Основная ее часть представляет собой запоминающее устройство, в котором информацию можно без каких-либо ограничений считывать и записывать. Такой вид памяти принято называть *оперативным запоминающим устройством* (ОЗУ). Соответствующий по смыслу английский термин, который часто встречается в технической литературе, — RAM (Random Access Memory), т. е. память с произвольным доступом. Сразу поясним, что такой доступ подразумевает возможность получать данные из памяти по любым адресам в любом порядке.

Технологическая основа ОЗУ может быть различной. Самые первые ЭВМ имели память на ртутных линиях задержки или электронно-лучевых трубках. Эти экзотические устройства были быстро вытеснены запоминающими устройствами на магнитных сердечниках, которые получили широкое распространение и применялись довольно долго. Намагниченное состояние небольшого ферритового кольца, которое служило элементом для запоминания одного бита информации, соответствовало единице, ненамагничен-

ное — нулю. Наконец, развитие технологий интегральной микроэлектроники позволило изготовить компактную полупроводниковую память, которая в данный момент и применяется в компьютерах. Таким образом, память сейчас производится теми же методами, что и микропроцессоры; не случайно фирма Intel, разработавшая и создавшая первый в мире микропроцессор, была в то время известна выпуском самых компактных микросхем ОЗУ [78, 10]. Описанию устройства полупроводниковой памяти посвящен следующий раздел. Тем читателям, кто бы хотел поподробнее познакомиться с общими принципами хранения информации в запоминающих устройствах, можно порекомендовать книгу [57].

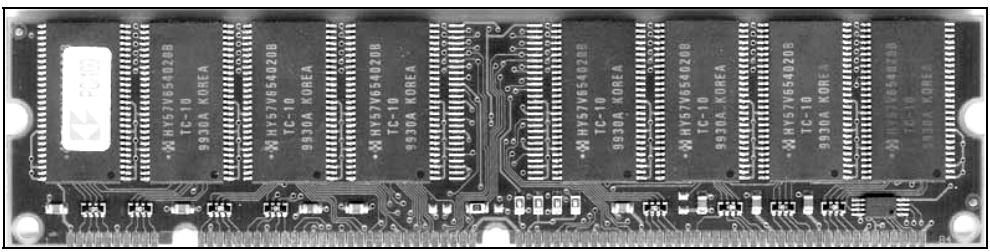


Рис. 4.1. Модуль с микросхемами ОЗУ

Так что же все-таки общего между всеми перечисленными конструкциями ОЗУ? Во-первых, возможность считывать и записывать информацию из произвольного места памяти (сравните с магнитной лентой, где информация может считываться только последовательно). Во-вторых, высокая (по сравнению с другими видами памяти) скорость работы ОЗУ, приближающаяся к быстродействию микропроцессора. И, наконец, в-третьих, необходимость специальных мер по сохранению информации из ОЗУ после завершения работы.

Последнее свойство иногда не совсем точно формулируют по-другому: при выключении питания информация ОЗУ теряется. На самом деле, это слишком сильное утверждение. Начать с того, что магнитное ОЗУ не требовало питания при хранении данных. Это замечательное свойство неоднократно использовалось автором и его коллегами при проведении продолжительных многодневных расчетов на старых добрых ЭВМ второго поколения следующим образом. Вечером машина с пульта управления останавливалась, на листочек записывалось содержимое программного счетчика и иногда некоторая другая служебная информация, высвечиваемая на том же пульте неоновыми лампочками. Затем после нажатия клавиши блокировки, предотвращавшей возможность всякой случайной записи в ОЗУ, машина выключалась. Вся информация в магнитной памяти прекрасно сохранялась до утра, когда ЭВМ снова включалась и программа запускалась с записанного накануне значения программного счетчика.

Примечание

Данный пример интересен не только как доказательство возможности сохранения содержимого ОЗУ после выключения питания. Он еще демонстрирует, как раньше решалась проблема задания начального адреса *(см. разд. 3.2.2)*: стартовый адрес задавался не автоматически как сейчас, а с пульта управления оператором по своему разумению.

Если кого-то не убедил предыдущий "ностальгический" пример, добавлю другой, более поздний. Замечательный по надежности изготовления японский учебный компьютер Yamaha, который не так давно широко применялся в нашей системе образования и без сомнения относился к четвертому поколению, легко позволял проделать над собой следующий эксперимент. После ввода программы в память, компьютер можно было выключить на 30—40 секунд и затем после включения, изменив содержимое пары определенных ячеек ОЗУ, продолжать работу с программой как ни в чем не бывало. Не будем детально анализировать последний пример (существенную роль в нем играли отдельные детали: качество блока питания и некоторые особенности загрузки программного обеспечения при включении машины), лишь сделаем вывод о том, что говорить о немедленном и полном исчезновении информации в ОЗУ при выключении питания можно только с некоторыми оговорками.

Другим важным видом внутренней памяти компьютера является *постоянное запоминающее устройство* (ПЗУ). Техническое (английское) название этого устройства памяти — ROM (Read Only Memory), т. е. память только для чтения. И действительно, ее содержимое можно только читать: никакая работающая программа не может изменить записанную там информацию, поэтому последняя всегда неизменна и постоянно доступна компьютеру, в том числе сразу в момент включения.

ПЗУ играет в современных компьютерах очень важную роль. Прежде всего, каждый компьютер содержит ПЗУ с программой начальной загрузки, о котором мы уже говорили. В этой же самой микросхеме обычно хранятся минимальные программы работы с клавиатурой и другими устройствами, поэтому ее часто называют *BIOS* — Basic Input/Output System (данные программы можно сравнить с врожденными безусловными рефлексами у живого существа). Кроме того, в ПЗУ может помещаться информация, необходимая для работы внешних устройств, например о начертании символов: так называемые знакогенераторы, которые были в каждом матричном принтере. Наконец, многие современные платы управления внешними устройствами, например, жестким диском, фактически являются настоящими специализированными ЭВМ и содержат собственное ПЗУ со встроенным программным обеспечением.

Техника формирования содержимого ПЗУ может быть различной. Первоначально это делалось только на заводе в процессе изготовления микросхемы. Затем появились постоянные запоминающие устройства, которые потреби-

тель мог заполнить сам, поместив "чистую" микросхему в специальное устройство, называемое *программатором*. Хорошее представление о подобных ПЗУ дают микросхемы, каждый бит памяти в которых моделировался тончайшей токопроводящей перемычкой: ее наличие означало единицу, а отсутствие — ноль. Перемычки могли пережигаться в программаторе, подобно тому, как сжигает ниточку предохранителя в бытовой аппаратуре недопустимый ток. Очевидно, что такой процесс формирования нулевых битов был необратимым.

Позднее появились конструкции ПЗУ, которые можно было запрограммировать несколько раз. Они получили название перепрограммируемые ПЗУ (ППЗУ). Стирание старой информации, т. е. установка всех битов памяти в единичное состояние, производилось ультрафиолетовым излучением (интенсивности бытового прибора для загара вполне хватало!), для которого в корпусе микросхемы делалось специальное прозрачное окошечко. Наконец, самые последние разработки ППЗУ позволяют производить обновление информации чисто электрическим путем, причем уже без специального программатора и не вынимая из платы. Одна из современных разновидностей ППЗУ такого рода называется *флэш-памятью* (flash memory). Стирание и запись во флэш-память происходит значительно медленнее, чем чтение из нее, тем не менее, в настоящее время она начинает использоваться в качестве внешней памяти как альтернатива дисковой (рис. 4.2). Любопытно заметить, что применение микросхем флэш-памяти в качестве BIOS без специальной блокировки записи (что на практике иногда встречается при неквалифицированной сборке компьютеров) может привести к попаданию туда

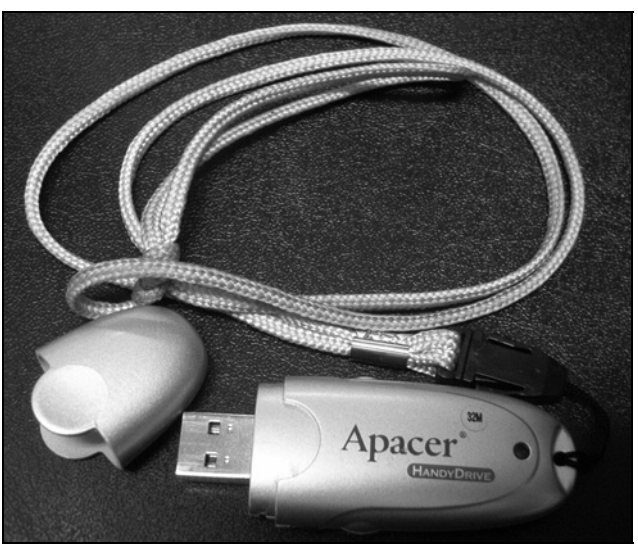


Рис. 4.2. Flash-память, организованная в виде миниатюрного диска

кода вируса; "лечение" в этом случае в домашних условиях будет практически невозможным.

Но и это еще не все виды памяти! В любом компьютере имеется особая память, питающаяся от отдельной батарейки, в которой хранится время, число, данные о конфигурации компьютера и т. д. Обо всех упомянутых и неупомянутых видах памяти довольно подробно и понятно написано в энциклопедии [15].

4.1.3. Статическое и динамическое ОЗУ

Хотя читатели, наверно, уже немного устали от всевозможных типов памяти, никак нельзя обойти вниманием еще два. Речь идет о том, что современные полупроводниковые микросхемы ОЗУ бывают двух видов — *статические* и *динамические*.

Базовым элементом статический памяти служит наш "старый знакомый" — триггер. Одно из двух его устойчивых состояний принимается за логический ноль, а другое — за единицу. Состояния действительно настолько устойчивы, что при отсутствии внешних воздействий (правда, только при подключенном напряжении питания!) могут сохраняться сколь угодно долго (подробную информацию о работе триггера см. в *разд. 4.6.5*). Динамические элементы памяти, напротив, не обладают таким свойством. Они фактически представляют собой конденсатор, образованный средствами полупроводниковых технологий; с некоторым упрощением можно сказать, что логической единице соответствует заряженный конденсатор, а нулю — незаряженный. Существенным недостатком динамической памяти является наличие постепенного самопроизвольного разряда конденсаторов через внешние схемы, что ведет к потере информации. Чтобы этого не происходило, конденсаторы динамической памяти необходимо периодически подзаряжать (такой процесс принято называть *регенерацией ОЗУ*).

Оба вида запоминающих микросхем успешно конкурируют между собой, поскольку ни одна из них не является идеальной. С одной стороны, статическая память значительно проще в эксплуатации, т. к. не требует регенерации, и приближается по быстродействию к процессорным микросхемам. С другой стороны, она имеет меньший информационный объем и бóльшую стоимость (в самом деле, изготовление конденсатора значительно проще, чем триггерной схемы, и требует на кремниевой пластине гораздо меньше места), сильнее нагревается при работе. На практике в настоящий момент выбор микросхем для построения ОЗУ решается в пользу динамической памяти.

Технология производства полупроводниковой памяти постоянно совершенствуется, поэтому с точки зрения конструкции и внутреннего устройства существует большое разнообразие микросхем памяти. Но мы с вами договорились не погружаться в быстро меняющиеся технические детали. А в их

быстрой смене нет никаких сомнений: попробуйте, например, сейчас найти новую память для компьютеров, выпущенных пару лет назад, и вы убедитесь в этом. А между тем машины еще вполне работоспособны и продолжают эксплуатироваться!

4.1.4. Кэш-память

Из предыдущего раздела стало ясно, что существует некоторое противоречие между быстродействующей, но дорогой статической памятью и худшей по характеристикам, но более дешевой, динамической. Разумным компромиссом для построения экономичных и производительных систем является использование промежуточной *кэш-памяти*.

Этот вид памяти заслуживает отдельного рассмотрения. Он появился относительно недавно, но, начиная с 486-го процессора, без кэш-памяти не обходится ни одна модель (не случайно в *разд. 3.2.4* кэширование уже упоминалось среди методов ускорения работы процессора). Название "кэш" происходит от английского слова "cache", которое обозначает тайник или замаскированный склад (в частности, этим словом называют провиант, оставленный экспедицией для обратного пути, или запас продуктов, например зерна или меда, который животные создают на зиму). "Секретность" кэша заключается в том, что он невидим для пользователя и данные, хранящиеся там, недоступны для прикладного программного обеспечения.

Кэш представляет собой "быструю" статическую память небольшого объема, которая служит для ускорения доступа к полному объему "медленной" динамической памяти. Основная идея работы кэш-памяти заключается в том, что извлеченные из ОЗУ данные или команды программы копируются в кэш; одновременно в специальный каталог адресов, который находится в той же самой памяти, запоминается, откуда информация была извлечена. Если данные потребуются повторно, то уже не надо будет терять время на обращение к ОЗУ — их можно получить из кэш-памяти значительно быстрее. Поскольку объем кэша существенно меньше объема оперативной памяти, его контроллер (управляющая схема) тщательно следит за тем, какие данные следует сохранять в кэш, а какие заменять: удаляется та информация, которая используется реже или совсем не используется. Контроллер также обеспечивает своевременную запись измененных данных из кэша обратно в основное ОЗУ. Мы не будем здесь рассматривать подробности этого процесса, заметим только, что организация записи технически сложнее, чем чтения, и поэтому поддержание постоянного соответствия между содержимым кэша и ОЗУ является предметом особой заботы инженеров.

В современных компьютерах кэш-память обычно реализуется по двухуровневой схеме. При этом первичный кэш (level 1 — уровень 1) встроен непосредственно внутрь процессора, а вторичный (level 2) устанавливается на системной плате. Как и для ОЗУ, увеличение объема кэша повышает эф-

фективность работы компьютерной системы. В этом легко убедиться на практике, если в настройках вашего компьютера предусмотрена возможность отключения кэша *(подробную информацию см. в разд. 4.6.4)*.

4.2. Организация внутренней памяти

4.2.1. Ячейка, слово, байт

Для того чтобы лучше понять принципы организации памяти современных ЭВМ, стоит кратко рассмотреть вопрос в историческом плане.

Наиболее просто была организована память в ЭВМ первых двух поколений. Она состояла из отдельных ячеек, каждая из которых считывалась или записывалась как единое целое. Любая ячейка имела свой номер (адрес); очевидно, что адреса соседних ячеек были последовательными целыми числами, т. е. отличались на единицу. В первых ЭВМ использовались данные только одного типа — вещественные числа, причем их длина из соображений простоты устройства обычно равнялась длине машинной команды и совпадала с разрядностью памяти и всех остальных устройств машины. Иными словами, размер ячеек ОЗУ и разрядность обработки данных в то время были постоянными и равными между собой. Напомним *(см. разд. 1.3.4)* попутно, что ячейка типичной ЭВМ этого времени состояла из 36 двоичных разрядов.

В ЭВМ третьего поколения идеология построения памяти существенно изменилась: минимальная порция информации для обмена с ОЗУ была установлена равной 8 двоичным разрядам, т. е. 1 байту. Введение байтовой структуры памяти сделало возможным обрабатывать несколько типов данных разной длины: символы текста — 1 байт, целые числа — 2 байта, вещественные числа обычной или двойной точности — 4 или 8 байт соответственно. Вместо термина ячейка был принят другой — *машинное слово*: слово равнялось 4 байтам и соответствовало длине стандартного вещественного числа. Все объемы информации стали измеряться в кратных единицах: двойное слово, полуслово и т. п.

Размер машинного слова был, по-видимому, выбран исходя из форматов информации (т. е. определялся объемом обрабатываемой информации), а не в связи с разрядностью каких-либо устройств ЭВМ. Для подтверждения этого приведем некоторые данные о структуре типичных машин третьего поколения из семейства ЕС [19]. АЛУ ЕС-1022 имело 16 двоичных разрядов, ЕС-1033 — 32 разряда, а ЕС-1050 — 64 разряда. В то же время за одно обращение к ОЗУ в ЕС-1022 и ЕС-1033 выбиралось 4 байта, в ЕС-1050 — 8 байт, а в ЕС-1045 — 16 байт. Таким образом, разнообразие цифр свидетельствует, что 32 разряда (4 байта) не являлись каким-то технически выделенным объемом информации.

Естественно, что адрес в машинах с байтовой организацией стал относиться к отдельному байту, и байты памяти получили возрастающие на единицу номера. В качестве адреса слова, которое состоит из нескольких последовательно расположенных байтов, удобно использовать адрес байта с наименьшим номером. В итоге адреса слов уже не будут меняться через единицу: приращение зависит от длины машинного слова в байтах (для случая ЕС ЭВМ оно, как вы, конечно, догадались, равнялось четырем).

В машинах третьего поколения появилось и еще несколько особенностей: разная длина команд, наличие специальной сверхоперативной регистровой памяти, вычисление эффективного адреса ОЗУ как суммы нескольких регистров и т. п. Все это получило дальнейшее развитие в микроЭВМ четвертого поколения.

Для современных компьютеров разрядность микропроцессора стала одной из важнейших характеристик, которая оказывает влияние в том числе и на организацию памяти. Она по-прежнему сохранила байтовый характер, зато, как только разрядность микропроцессоров достигла нескольких байт, к машинному слову снова постепенно вернулась связь с аппаратным устройством машины. Стали говорить о 16- или 32-разрядном машинном слове в зависимости от разрядности регистров микропроцессора.

Важной особенностью микрокомпьютеров является также существенное усовершенствование методов адресации к информации в памяти, что будет более подробно рассмотрено позднее в *разд. 4.3*.

В большинстве книг явно сказано, что использованные нами в этом разделе понятия "ячейка памяти" и "машинное слово" являются синонимами. Вот как выглядят наиболее строгие определения, которые мне удалось найти [56]:

> "Ячейка памяти — вместилище порции информации в *памяти* ЭВМ, доступной для обработки отдельной командой. Обычно имеет фиксированный для данной ЭВМ размер... Содержимое Я. памяти называется *машинным словом*".
>
> "Слово в информатике — цепочка *битов*..., стандартно обрабатываемая как единый элемент *данных*. Применительно к оборудованию под С. понимают данные, занимающие ячейку памяти и позволяющие их использовать в качестве аргументов команды. Термин "С." употребляется как синоним термина "ячейка", а также в производных терминах "полуслово", "двойное С."..."

В порядке обсуждения последнего тезиса хочется добавить следующее. По мнению автора, классические определения ячейки и слова применительно к современным микропроцессорам выглядят не очень убедительно (не говоря о том, что нехорошо определять два термина друг через друга). Интуитивно хотелось бы видеть в размере ячейки памяти какую-то характеристику ОЗУ (например, почему бы не считать ячейкой памяти 1 байт — в конце концов

адрес памяти определен именно для байта!), а в длине машинного слова — особенность обрабатывающего информацию процессора, причем в обоих случаях их величины должны быть как-то связаны со внутренним устройством компьютера. Между прочим, в том, что разрядность ОЗУ и процессора не всегда одно и то же, мы уже убедились в *разд. 3.1.5*. Если вас, уважаемый читатель, заинтересовала данная проблема, обязательно рассмотрите вопрос 8 *в разд. 4.5*.

4.2.2. О хранении многобайтовых данных

Одного байта во многих случаях недостаточно для хранения какой-либо информации. Например, согласно старым стандартам под целое число требовалось 2 байта; в современных компьютерах часто необходимы большие значения, поэтому целое число может занимать 4 или даже 8 байт! Следовательно, надо иметь возможность сохранять информацию в нескольких соседних байтах.

Чтобы расположить информацию в нескольких байтах памяти, можно выбрать один из двух следующих способов (рис. 4.3).

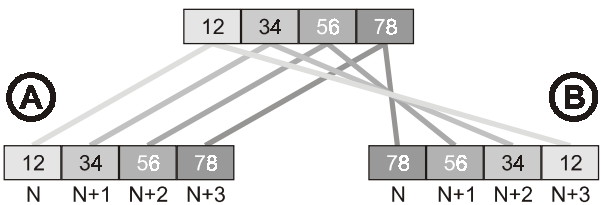


Рис. 4.3. Два способа хранения многобайтовых данных

Таблица 4.1. *Способы хранения многобайтовых данных*

A. Байт с наиболее значащей частью ("big-end", в исходном числе он находится слева) сохраняется в память по наименьшему адресу (на рисунке это **N**). Такой способ принято называть *"big-endian"* — по-русски говорят "прямое размещение байтов". Попутно заметим, что наиболее значащий байт числа в компьютерной литературе часто обозначают MSB (Most Significant Byte).	**B.** Байт с наиболее значащей частью (слева) сохраняется в память по наибольшему адресу (на рисунке это **N+3**). Следовательно, первым, наоборот, сохраняется байт с наименьшей значащей частью ("little-end", в исходном числе находится справа). Такой способ принято называть *"little-endian"* — по-русски говорят "обратное размещение байтов".

Из табл. 4.1 следует, что различие состоит в том, "с какого конца" (по-английски "end") необходимо начинать сохранять многобайтовое данное.

Вот отсюда-то и происходит сам образный термин. Он был предложен в одной из статей, посвященной рассматриваемому вопросу со ссылкой на книгу Джонатана Свифта "Приключения Гулливера". Как вы, наверно, помните, там ради обсуждения проблемы с какого конца — тупого или острого (по-английски "big side" или "little side") разбивать яйцо, были созданы две непримиримые политические партии. Во многом компьютерные дебаты по обсуждению преимуществ big- или little-endian выглядят аналогично.

Распространенные у нас компьютеры с Intel-совместимыми процессорами используют архитектуру "little-endian"; аналогичный способ был принят также в машинах PDP-11 и VAX. Таким образом, в нашей стране это наиболее известный метод хранения данных. Тем не менее, существовали и до сих пор существуют "big-endian" компьютеры, например, IBM 370, Motorola 68000 (семейство компьютеров Apple), Sun Sparc и многие RISC-процессоры. А вот система PowerPC "понимает" сразу оба формата данных и ее иногда называют *"bi-endian"*.

Отметим, что существенным является не столько то, как компьютерная система хранит данные внутри себя (в конце концов, это ее внутреннее дело), а то, как она их сохраняет "снаружи" в файлах. Это с практической точки зрения гораздо важнее.

Рис. 4.4. Диалоговое окно о порядке сохранения байтов в программе

В качестве примера представим, что текст "UNIX" в качестве двух 2-байтовых слов сохранен "big-endian" системой. Тогда "little-endian" система расшифрует его как "NUXI", переставив буквы в каждом из машинных слов. Это "недоразумение", также связанное с обсуждаемым вопросом, иногда называют "проблемой NUXI". Аналогичные трудности могут возникать и при сохранении графических изображений, поскольку цвета в современных компьютерах тоже кодируются многобайтовыми числами. Так, например, широко распространенный фотографический формат JPEG использует схему "big-endian", а не менее распространенные файлы GIF и BMP — "little-endian" (более подробный перечень файлов можно посмотреть в [И-28]). Иногда решение о порядке байт в файле программа предоставляет пользователю. На рис. 4.4 изображено диалоговое окно по этому вопросу в ходе сохранения файла (программа Adobe PhotoShop): byte order — это и есть тот самый порядок байтов, который мы только что обсуждали.

4.3. Адресация памяти

4.3.1. Адресное пространство памяти

Обсуждая устройство процессоров в *разд. 3.1.5*, мы видели, что каждый из них имеет максимальное адресуемое пространство. Оно зависит от количества двоичных разрядов шины адреса. По мере роста потребностей в компьютерной памяти разрядность постоянно увеличивают, но по прошествии какого-то времени памяти снова не хватает и все повторяется.

Не следует думать, что обсуждаемая характеристика ставит принципиальное ограничение на количество памяти. Так, многие 16-разрядные компьютеры со стандартным адресным пространством 64 Кбайт имели 128 Кбайт памяти и даже более. Секрет в том, что блоки памяти (их часто называют страницами) подключаются по очереди. Само переключение страниц памяти осуществляется аппаратно, но управление им всегда реализуется программным путем. Иными словами, программа способна подключить нужную страницу памяти и произвести с ней необходимый обмен информацией.

Не стоит впадать и в другую крайность, считая, что все адресное пространство обязательно должно быть заполнено. Сопоставьте типичные значения объема памяти современного компьютера с максимально адресуемым пространством процессора Pentium, и вы легко поймете, что это не так.

4.3.2. Методы адресации данных

Мы уже знаем, что при обмене информацией с памятью процессор обращается к ячейкам ОЗУ по их адресам. Способы задания требуемых адресов в командах ЭВМ принято называть *методами адресации*. От видов и разнообразия методов адресации существенно зависит эффективность работы с данными.

В первых ЭВМ адреса ячеек ОЗУ входили непосредственно в команду (вспомните разговор об адресности команд процессора в *разд. 3.3.3*). В современных компьютерах в самой команде адрес, как правило, явным образом не указывается, т. к. при подобном способе слишком возросла бы ее длина. Чтобы избежать этого, при обращении к ОЗУ процессор использует *метод косвенной адресации*. Его идея состоит в том, что адрес памяти предварительно заносится в один из регистров процессора, а команда содержит лишь ссылку на этот регистр. Если еще учесть, что адреса в регистре очень удобно модифицировать (скажем, циклически увеличивая на заданную величину), станет понятным, почему косвенная адресация нашла такое широкое применение.

Кратко опишем наиболее распространенные варианты ссылок на данные в машинной команде (учитывая, что терминология для разных микропроцессоров может отличаться, названия методов адресации не приводятся).

1. Данные находятся в одном из регистров МП.
2. Данные входят непосредственно в состав команды, т. е. размещаются внутри ее кода (константа).
3. Данные находятся в ячейке ОЗУ, адрес которой содержится в одном из регистров МП.
4. Данные находятся в ячейке ОЗУ, адрес которой вычисляется по формуле:

   базовый адрес + *смещение*

   Базовый адрес хранится в одном из регистров МП и является начальной точкой массива данных. Смещение может быть как некоторой константой, так и содержимым другого регистра.

Следует подчеркнуть, что здесь описаны лишь наиболее общие методы адресации. У конкретных моделей ЭВМ существуют некоторые особенности адресации ОЗУ. Так, например, в процессорах семейства PDP возможна двойная косвенная адресация: данные могут храниться в ячейке ОЗУ, адрес которой берется из другой ячейки, адрес которой, в свою очередь, находится в указанном регистре (как тут не вспомнить детский стишок о синице, которая ворует пшеницу, которая в темном чулане хранится, в доме, который построил Джек?).

Методы адресации могут быть и более экзотическими, например, широко распространенный сегментный способ процессоров фирмы Intel. В одной из своих популярных книг [62] Питер Нортон метко назвал этот способ словом "клудж" (по-английски "kludge" — приспособление для временного устранения проблемы). Сегментный метод был предложен для первого 16-разрядного процессора 8086 для того, чтобы можно было получить в нем 20-разрядный адрес и тем самым расширить максимально возможный объем ОЗУ. Суть метода состоит в том, что адрес вычисляется как сумма двух чи-

сел — сегмента и смещения, причем одно из них сдвинуто влево на 4 двоичных разряда. Пусть, например, сегмент в шестнадцатеричном виде равен A000, а смещение — 1000. Тогда итоговый адрес равен:

$$\begin{array}{r} A\ 0\ 0\ 0(0) \\ +\ \underline{(0)1\ 0\ 0\ 0} \\ A\ 1\ 0\ 0\ 0 \end{array}$$

"Такой способ адресации более 64 Кбайт памяти кажется довольно странным, однако он работает", — не без некоторой иронии замечает П. Нортон.

К счастью, современные "интеловские" процессоры, начиная с 80386, стали 32-разрядными, и их регистров хватает, чтобы адресовать до 4 Гбайт памяти. Сегментный способ при этом становится излишним, хотя и сохраняется ради обеспечения программной совместимости с предыдущими моделями. Между прочим, в некоторых наших экспериментах *(см. разд. 2.5.2 и 4.6.3)* нам "по старинке" приходится пользоваться именно этим методом доступа к байтам ОЗУ.

4.3.3. Стековая организация памяти

Нам осталось рассмотреть еще один очень специфический, но интересный и часто используемый на практике способ адресации данных в ОЗУ. Речь пойдет о работе со стеком. *Стек* — это неявный способ адресации, когда информация хранится в виде последовательности, в которой доступен только последний элемент. Хорошей аналогией стека может служить детская пирамидка, нижнее кольцо которой невозможно снять, пока не будут сняты все остальные.

Обратимся к примеру. Пусть требуется на время сохранить значения трех целочисленных 2-байтовых переменных `N1`, `N2` и `N3`, а позднее их все восстановить. Воспользуемся для этого стековой памятью. Прежде всего, запомним, что стек всегда имеет единственный вход и выход информации — для хранения его адреса служит специальный регистр микропроцессора под названием *указатель стека.* Пусть для определенности он сейчас содержит адрес 2006 (рис. 4.5, *а*). Тогда по команде "записать в стек `N1`" (обратите внимание, что в команде не фигурирует явно ни адрес ОЗУ, ни регистр указателя стека!) процессор проделает следующее:

- уменьшит указатель стека на 2;
- запишет `N1` по полученному адресу 2004 (рис. 4.5, *б*).

Аналогично при выполнении команд "записать в стек `N2`" и "записать в стек `N3`", значения этих переменных попадут в ячейки 2002 и 2000, причем указатель стека станет равным 2000 (рис. 4.5, *в*).

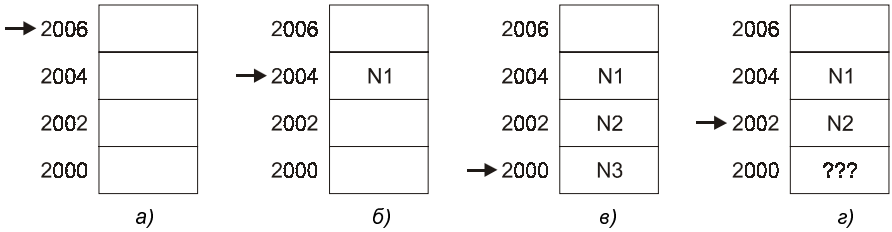


Рис. 4.5. Пример работы со стеком

Теперь займемся извлечением информации. Выполним команду "прочитать из стека в N3". При этом процессор проделает следующее:

- считает в N3 значение из стека;
- увеличит указатель на 2 (рис. 4.5, *г*).

Аналогично прочитаем N2 и N1, после чего стек опустеет и вернется к начальному состоянию, изображенному на рис. 4.5, *а*.

Примечание

Обратим внимание на то, что значение в стеке после считывания, конечно же, не исчезает, как это условно показано на рис. 4.5, *г*, но есть причины полагать, что мы его там можем больше не увидеть. Дело в том, что процессор иногда временно использует стек для своих "внутренних" нужд (подробности будут описаны при рассмотрении работы с внешними устройствами). При этом некоторые ячейки, адреса которых меньше текущего указателя стека, естественно, изменятся. Следовательно, во избежание неприятностей лучше всегда считать, что после считывания информации в стековой памяти она пропадает.

Подводя итоги рассмотрения примера, подчеркнем, что информация, которая заносится в стек первой, извлекается последней, и наоборот. Обычно об организации стека образно говорят: "первым пришел — последним ушел".

Мы видели, что в командах работы со стеком адрес ОЗУ не фигурирует в явном виде. Но при этом молчаливо предполагалось, что указатель стека уже установлен на свободную область ОЗУ — в противном случае запись в стек может стереть нужную информацию. Ответственность за правильность значения указателя стека лежит на программном обеспечении.

Интересно, что со стековым способом организации мы имеем дело в жизни гораздо чаще, чем это может показаться на первый взгляд. Во-первых, дисциплине "первым пришел — последним ушел" подчиняются скобки в арифметических выражениях: первой всегда закрывается "самая внутренняя", т. е. как раз последняя открытая скобка. Если вы знакомы с программированием на любом алгоритмическом языке, то без труда найдете и еще два примера: вложенные циклы и подпрограммы. Справедливость этого утверждения предлагаю обдумать самостоятельно.

Остается сказать, что стек используется в вычислительной технике очень широко. На аппаратном уровне процессор "обучен" запоминать в стеке текущее значение счетчика адреса команд при вызове подпрограммы. Часто в тот же самый стек предварительно заносятся необходимые для подпрограммы параметры: так реализуется, например, вызов процедур и функций с параметрами в языке Паскаль. Кроме того, стек очень часто используется для временного сохранения значений тех или иных внутренних регистров процессора. Наконец, процессор активно использует стек при реализации прерываний от внешних устройств, о чем мы подробнее поговорим позднее.

4.4. Выводы

Перейдем к выводам.

- Память — это одно из основных устройств ЭВМ, которое служит для хранения программы и обрабатываемых данных.
- Деление памяти на внутреннюю и внешнюю хотя и имеет историческое происхождение, но сохраняется до сих пор.
- Внутренняя память может строиться на самых разнообразных физических принципах, но по функциональному назначению различают ОЗУ, ПЗУ, ППЗУ и некоторые другие виды памяти.
- Современные ОЗУ изготовляются в виде полупроводниковых микросхем двух типов — статических и динамических. Каждый из них имеет свои достоинства и недостатки. В качестве компромиссного решения служит применение кэш-памяти.
- Внутренняя организация памяти с развитием вычислительной техники претерпела значительные изменения. Из-за необходимости хранения и обработки данных различной длины произошел переход от многоразрядных ячеек фиксированной длины к более гибкой байтовой структуре.
- При сохранении в память данных, содержащих несколько байт, возможно два способа: по сложившейся традиции их называют "little-endian" и "big-endian". Большинство используемых в нашей стране машин применяют первый способ.
- Для того чтобы указать в команде адреса необходимых ячеек памяти, существуют различные методы адресации данных. Одна из наиболее принципиальных идей в этой области состоит в том, что адрес памяти помещают в регистр микропроцессора, а в команде содержится ссылка на этот регистр (косвенная адресация).
- Одним из важных способов организации памяти в современных компьютерах является стек. Он используется при вызове подпрограмм, временном хранении данных и для передачи параметров.

4.5. Вопросы для осмысления

1. Каковы характерные черты ОЗУ и ПЗУ? Какие еще виды памяти вы знаете?
2. Подумайте, какую информацию вы бы занесли в ПЗУ компьютера.
3. Что такое BIOS и каковы его функции?
4. Перечислите достоинства и недостатки статических и динамических микросхем ОЗУ. Какие из них в современных компьютерах используются чаще? Почему?
5. Что такое кэш-память и как она работает? Объясните, как кэш ускоряет работу процессора.
6. Можно ли считать из памяти один отдельно взятый бит? Какой логической операцией можно выделить необходимый бит считанного кода?
7. Как вы понимаете термины "машинное слово" и "ячейка памяти"?
8. Известно [73], что процессор Intel может считывать 2-байтовые слова с нечетным адресом. Для этого он последовательно извлекает два соседних слова с четными адресами, а затем из каждого берется по одному байту. Попробуйте применить к данному примеру понятия "ячейка" и "слово".
9. Что такое "проблема NUXI"? Приведите еще два-три подобных примера изменения слов (кстати, сможете ли вы подобрать такой, чтобы и исходное, и полученное слово были осмысленными?) Как будет проявляться данная проблема, если машина не 16-, а 32-разрядная?
10. Оцените, какой процент адресного пространства вашего компьютера реально заполнен памятью.
11. Какие методы адресации данных вы знаете? Какой из них удобно применить для обращения к элементам массива?
12. Используя команды записи и чтения из стека, описанные в *разд. 4.3.3*, составьте программу обмена значениями между двумя переменными.

4.6. Любопытные эксперименты

4.6.1. Сколько в компьютере ОЗУ?

Этот эксперимент во многом похож на то, что мы делали с целью определения марки процессора *(см. разд. 3.6.1)*.

Цель эксперимента. Научиться определять количество оперативной памяти в компьютере.

Первая информация о количестве ОЗУ появляется почти сразу после включения компьютера: одним из тестов, которые машина выполняет для само-

проверки, является тест памяти, в ходе прохождения которого на экране, как правило, отображается объем проверенной памяти.

Примечание

В некоторых компьютерах тест памяти может быть отключен с помощью программы настройки (так называемой BIOS Setup). К сожалению, вид этой программы существенно зависит от производителя компьютера, поэтому трудно привести какие-либо универсальные рекомендации по ее использованию. В своем изложении автор предполагает, что вы уже немного умеете пользоваться BIOS Setup своей машины. Если это не так, то можно попытаться посмотреть прилагаемую к компьютеру документацию: хорошая компьютерная фирма обязательно кладет в коробку небольшую книжечку на русском языке, где в одном из разделов подробно и с многочисленными иллюстрациями описано, как менять настройки компьютера с помощью этой важной "встроенной" программы. Наконец, всегда можно проконсультироваться у своих более знающих друзей.

Для того, чтобы изучать содержимое экрана при загрузке внимательно и не торопясь, не забывайте о возможности пользоваться клавишей <Pause>.

Точно так же, как мы это делали в эксперименте, описанном в *разд. 3.6.1*, можно обратиться за информацией об объеме ОЗУ к операционной системе. Более того, сведения об оперативной памяти расположены на той же самой страничке, что и о процессоре, на что многие читатели уже обратили внимание, когда рассматривали рис. 3.4.

Для более детального изучения состава ОЗУ в своем компьютере можно использовать специальные программы-утилиты. Например, уже упоминавшаяся ранее утилита CPU-Z [И-22], помимо своего прямого назначения — отображения информации о центральном процессоре, способна анализировать и состав ОЗУ: из рис. 4.6 видно, что общая память в домашнем компьютере автора составляет 160 Мбайт, причем состоит она из двух микросхем — 128 и 32 Мбайт (последняя когда-то была единственной).

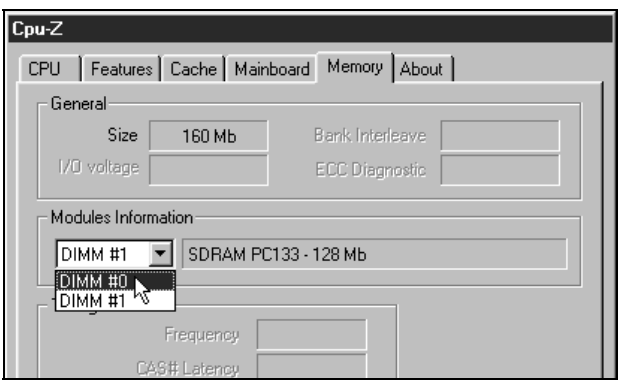


Рис. 4.6. Анализ состава ОЗУ

Исследуя ОЗУ своего компьютера, обратите внимание на то, что микросхемы памяти всегда имеют объем, равный степени двойки (128 и 32 в последнем примере). Если индицируется несколько меньшее значение, то, возможно, ваш компьютер некоторый объем ОЗУ использует в качестве видеопамяти. Например, операционная система "рапортует" о 120 Мбайт ОЗУ, а "недостающие" 8 Мбайт подключены как видеопамять.

Вывод. Таким образом, получить наиболее общую информацию об объеме ОЗУ не представляет труда.

4.6.2. Как используется ОЗУ?

В предыдущем эксперименте мы узнали объем ОЗУ в своем компьютере. В связи с полученным результатом немедленно возникает вопрос, а достаточно ли такого объема. Для изучения этого вопроса — новый эксперимент.

Цель эксперимента. Контролировать, как расходуется память компьютера в процессе работы.

Распределением памяти компьютера занимается операционная система, следовательно, к ней и надо обратиться за справкой по интересующей нас проблеме. В операционных системах для этой цели обычно имеется специальная утилита: в MS-DOS она называлась MEM, в Windows 98 — системный монитор, а в Windows 2000 — диспетчер задач. Первая выдавала информацию в виде текстовых справок, а остальные, согласно изменившимся общим принципам организации интерфейса, рисуют цветные графики. Здесь мы рассмотрим, как использовать для контроля системный монитор. И хотя в других системах конкретные действия и режимы контроля могут несколько отличаться, основные принципы работы достаточно похожи.

Для запуска системного монитора требуется после нажатия кнопки **Пуск** проследовать через меню **Стандартные** в **Служебные**, где и должна находиться ссылка на данную программу. Возможно, среди имеющихся в данном пункте меню программ вы не увидите системного монитора: это означает, что он не был востребован при установке системы. Ситуация легко исправляется стандартным образом путем обновления состава компонентов Windows.

Итак, мы запустили системный монитор. Пользуясь меню **Правка** можно в ходе несложного диалога выбрать, за какими характеристиками системы мы будем наблюдать. Автор предлагает взять наиболее важные для нас сейчас параметры **Выделено памяти** и **Свободная физическая память**. После этого окно монитора будет выглядеть примерно так, как показано на рис. 4.7 (разумеется, масштаб по осям и форма кривой у каждого компьютера будут свои).

На рассматриваемом рисунке продемонстрированы изменения использования ОЗУ для нескольких типичных ситуаций. На участке графика, помечен-

ном на рис. 4.7 цифрой **1**, была запущена небольшая программа Corel Capture для "захвата" содержимого окна монитора. Затем в районе цифры **2** в демонстрационных целях был запущен мощный графический редактор CorelDraw! Отчетливо видно, что это потребовало существенно больше ресурсов, чем для предыдущей программы. Некоторое время (горизонтальный участок **3**) редактор ожидал выбора режима работы. После указания создать новый документ редактор запросил у системы еще некоторый небольшой объем памяти (цифра **4**). Наконец, последний ниспадающий участок **5** отражает закрытие редактора.

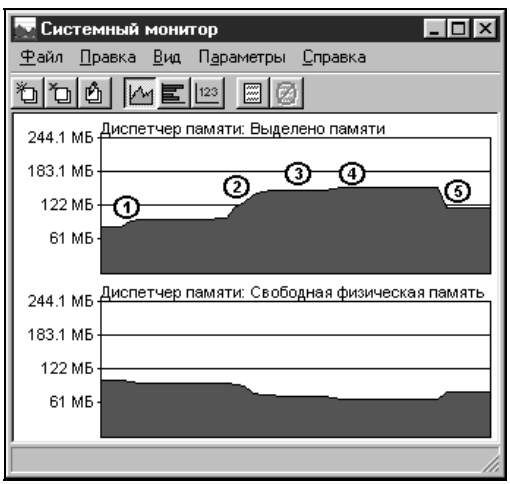


Рис. 4.7. Контроль использования памяти компьютера

Таким образом, контроль использования памяти оказывается простым и наглядным. Отчетливо видно, как система выделяет память, а свободная физическая память при этом уменьшается.

Обязательно обратите внимание на тот факт, что не вся память после закрытия приложения освобождается. Операционная система оставляет некоторые блоки кода приложения в памяти, что позволяет ускорить повторный запуск данной программы. К сожалению, постепенно происходит "замусоривание" памяти и свободная память уменьшается. Через 3—4 часа работы ее размер может при всех завершенных задачах стать значительно ниже, чем сразу после загрузки. Наблюдение за графиком свободной физической памяти позволяет оценить, достаточно ли ее установлено в вашем компьютере. Если график опускается практически до нуля, то памяти перестает хватать.

Примечание

Мне кажется, у читателя должен возникнуть естественный вопрос, что будет делать компьютер, когда объем свободной памяти упадет до нуля. Краткий от-

вет звучит обнадеживающе: ничего фатального не произойдет, хотя работа машины существенно замедлится. Подробности станут легко понятны после прочтения *разд. 5.3.2* следующей лекции.

Наблюдение за расходованием памяти довольно любопытное занятие и автор надеется, что многие читатели некоторое время последят за этим процессом при работе со своими любимыми приложениями.

Вывод. Операционная система предоставляет определенные средства контроля над расходом памяти в компьютере. Использование этих средств позволяет оценить, является ли количество установленной в компьютере памяти достаточным.

4.6.3. Порядок хранения данных в IBM PC

В лекции утверждается, что в IBM PC многобайтовые данные хранятся, начиная с младшего байта, т. е., выражаясь житейским языком, "задом наперед".

Цель эксперимента. Проверить, в каком порядке записаны данные в память IBM PC.

С целью проверки данного факта выберем десятичное число

$$513 = 2 * 256 + 1$$

Из приведенной записи видно, что старший байт числа равен 2, а младший — 1. Остается проверить, в каком порядке они лежат в памяти. Для этого используем следующую короткую программу (листинг 4.1).

Листинг 4.1. Программа проверки порядка хранения данных в IBM PC

```
PROGRAM endian;
VAR   i:INTEGER; b1,b2:BYTE;
BEGIN i:=513;
      b1:=MEM[SEG(i):OFS(i)];
      b2:=MEM[SEG(i):OFS(i)+1];
      WRITELN(i,' ',b1,' ',b2);
      IF i=256*b2+b1 THEN WRITELN('little endian')
                     ELSE WRITELN('big endian')
END.
```

Ее наиболее трудной частью является извлечение из памяти двух последовательных байт: `b1` и `b2`:

```
b1 := MEM[SEG(i):OFS(i)]; b2 := MEM[SEG(i):OFS(i)+1];
```

Функции `SEG()` и `OFS()` (их названия происходят от английских слов "segment" и "offset", т. е. "сегмент" и "смещение" соответственно) задают адрес указанного байта в стандарте, принятом в IBM PC *(см. пример в*

разд. 4.3.2). Как видно из формул, байт `b1` имеет меньший адрес. Предопределенный в Паскале массив `MEM` позволяет извлечь лежащий по указанному адресу ОЗУ байт.

Остальная часть программы значительно проще. Она выводит на экран полученные числовые значения, а затем сравнивает, действительно ли старший байт, который умножается на 256, лежит вторым (т. е. соответствует `b2`) в соответствии со стандартом "little-endian".

Запуск программы подтверждает правильность нашего предположения. Даже если написать любое другое положительное число, не превышающее 32 767, диагностическое сообщение останется прежним.

Вывод. В памяти IBM PC используется обратное ("little-endian") размещение байтов.

4.6.4. Оценка эффективности применения кэш-памяти

В любой книге по устройству компьютера сказано, что применение кэш-памяти ускоряет работу машины за счет сокращения времени доступа к содержимому ОЗУ. Данный эксперимент посвящен проверке этого теоретического утверждения. Кроме того, читателям, скорее всего, будет интересно, насколько ускоряется обработка информации в количественном отношении.

Цель эксперимента. Проверить, насколько эффективно ускорение работы компьютера благодаря использованию кэш-памяти.

Методика эксперимента проста: для того чтобы узнать, насколько было полезно то или иное устройство, надо его отключить. Возможность отключения кэш-памяти предусмотрена в BIOS Setup. Ранее уже отмечалось, что устройство этой программы на различных машинах неодинаково, но, как показал опыт, способ доступа к отключению кэша примерно одинаков. Для этого необходимо войти в программу и выбрать пункт **Advanced BIOS Features** (можно перевести как усовершенствованные или расширенные характеристики BIOS) и в нем искать строки, содержащие слова **Internal Cache** или **External Cache**, обозначающие внутренний или внешний кэш. Вполне возможно, что они будут содержать обозначения `L1` или `L2`, что означает уровень кэша. Как бы ни назывался этот параметр, значений у него всегда только два: по-английски они называются enabled (включено) или disabled (выключено). Клавиши, с помощью которых можно изменить значение выбранного параметра, обычно указаны в пояснениях на экране; могут применяться довольно разные клавиши, но автор все же надеется, что читателям удастся их определить и отключить кэш. Дальше остается только измерить время выполнения одной и той же программы с включенной и выключенной кэш-памятью. Результаты будут зависеть как от тестируемого компьютера, так и от самой тестовой программы.

Автор проверил несколько компьютеров и обнаружил хорошо заметную разницу между временем исполнения программы с кэшем и без оного. Так для компьютера с процессором Celeron с тактовой частотой 866 МГц выигрыш при использовании кэша составил около 20 %, а для Pentium 4 с частотой 1,4 ГГц эффект оказался значительно больше — почти 2,3 раза. Для сравнения аналогичные эксперименты были проведены со старым компьютером 80486DX2-66. Эффективность кэш-памяти получилась примерно 40 %.

Результат эксперимента зависит от того, какая программа в нем используется. Читатели могут проверить это утверждение самостоятельно.

Вывод. Эксперимент показывает, что применение кэш-памяти заметно ускоряют работу компьютера. Необходимо подчеркнуть, что приведенные цифры есть не более чем грубая оценка, которая ни в коем случае не претендует на далеко идущие обобщения.

4.6.5.* Экспериментальное изучение триггера

Мы уже неоднократно встречались в лекциях с электронным устройством, которое называется триггер. В данном разделе мы познакомимся с ним подробнее. Сам по себе эксперимент будет очень несложным, но потребует некоторой подготовки — придется изготовить триггер своими руками. Для этого будет необходимо подобрать очень небольшой набор радиодеталей и иметь некоторый навык работы с паяльником. Но даже если по какой-либо причине вы не станете повторять описанный здесь эксперимент, его теорию стоит прочитать в любом случае.

Цель эксперимента. Познакомиться с основными принципами устройства и функционирования триггера и проверить их на практике.

Краткие теоретические сведения.

Триггер — это электронная схема, которая может находиться в одном из двух устойчивых состояний: им условно приписывают значения 0 и 1. При отсутствии входных сигналов триггер способен сохранять свое состояние сколь угодно долго. Таким образом, из определения следует, что триггер способен хранить ровно 1 бит информации.

Можно без преувеличения сказать, что триггер является одним из существенных узлов при проектировании ЭВМ. Отдельно взятые триггеры используются довольно редко; как правило, некоторое количество триггеров объединяют вместе, при этом полученное устройство называется *регистром*. Регистры содержатся во всех вычислительных узлах, начиная с центрального процессора и кончая периферийными устройствами.

Очень важно подчеркнуть, что триггеры, объединенные в регистр, кроме способности хранения информации часто приобретают новые свойства, которые позволяют им ее обрабатывать. Широкое распространение получили, например, так называемые *сдвиговые регистры*, которые, как легко понять из

названия, предназначены для выполнения сдвига двоичного кода. Интересным и важным для практики объединением триггеров являются *счетчики* — схемы, способные считать поступающие на их вход импульсы. Так что, как видите, разнообразие схем на базе триггеров достаточно велико.

Рассмотрим логическое устройство триггера. На рис. 4.8 *а* приведена простейшая схема триггера, а на рис. 4.8 *б* показано его обозначение на схемах как единого функционального узла.

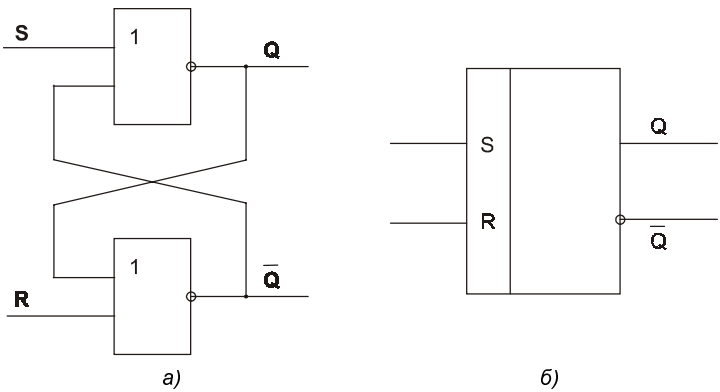


Рис. 4.8. Логическая схема триггера и его функциональное обозначение

Описание начнем с расшифровки обозначений входов и выходов. Триггер имеет два входа — **S** (от английского слова Set — установка) и **R** (Reset — сброс), которые используются соответственно для установки триггера в единичное и сброса в нулевое состояния. Вследствие таких обозначений рассматриваемую схему назвали *RS-триггером*. Один из выходов, обозначенный на схеме **Q**, называется *прямым*, а противоположный выход — *инверсным* (это индицирует черта над **Q**, которая в математической логике обозначает отрицание). Очевидно, что сигналы на прямом и инверсном выходах всегда противоположны. За единичное состояние триггера договорились принимать такое, при котором **Q** = 1.

Обратимся теперь к схеме на рис. 4.8, *а*. Видно, что триггер состоит из двух одинаковых двухвходовых логических элементов ИЛИ-НЕ (ИЛИ обозначается символом **1** внутри элемента, а отрицание НЕ — небольшим кружочком на его выходе), соединенных определенным симметричным образом. Сигнал на один из входов каждого элемента снимается с выхода другого. Именно наличие такого соединения и дает триггеру возможность сохранять свое состояние после прекращения действия сигналов.

Рассмотрим, как работает схема. Подробные рассуждения по этому поводу можно прочитать в методическом пособии [31]. Пусть для определенности на вход **S** подан единичный сигнал, а **R** = 0. Тогда независимо от состояния

другого входа, который подсоединен к выходу **Q** (иначе говоря, вне зависимости от предыдущего состояния триггера!), верхний по схеме элемент ИЛИ-НЕ получит на выходе 0 (результат ИЛИ, естественно, равен 1, но его инверсия — 0). Этот нулевой сигнал передается на вход другого логического элемента, где на втором входе **R** тоже установлен 0. В итоге после выполнения логических операций ИЛИ-НЕ над двумя входными нулями, этот элемент получает на выходе единицу, которую "возвращает" первому элементу на соответствующий вход. Последнее обстоятельство очень важно: теперь, когда на этом входе установилась 1, состояние другого входа (**S**) больше не играет роли. Иными словами, если даже теперь убрать входной сигнал **S**, внутреннее распределение уровней сохранится без изменения. Поскольку, согласно нашим рассуждениям **Q** = 1, триггер перешел в единичное состояние и, пока не придут новые внешние сигналы, сохраняет его. Итак, при подаче сигнала на вход **S** триггер переходит в устойчивое единичное состояние.

При противоположной комбинации сигналов **R** = 1 и **S** = 0 вследствие полной симметрии схемы все происходит совершенно аналогично, но теперь на выходе **Q** уже получается 0. Иными словами, при подаче сигнала на вход **R** триггер сбрасывается в устойчивое нулевое состояние.

Особо отметим, что окончание действия сигнала в обоих случаях приводит к тому, что **R** = 0 и **S** = 0. Мы видели, что при этом триггер сохраняет на выходе **Q** тот сигнал, который был установлен входным импульсом (**S** или **R**). Отсюда такой режим часто называют *режимом хранения информации*. Итак, при отсутствии входных сигналов триггер сохраняет последнее занесенное в него значение сколь угодно долго.

Оставшийся режим **S** = 1 и **R** = 1, когда сигналы подаются на оба входа одновременно, считается запрещенным, поскольку в этом случае после снятия входных сигналов (особенно одновременного!) результат непредсказуем. В более сложных триггерах подобная ситуация обрабатывается при помощи специальной входной логики.

Все проведенные выше рассуждения, как это обычно делается [51], удобно представить в виде следующей таблицы (табл. 4.2).

Таблица 4.2. *Таблица режимов триггера*

Вход S	Вход R	Выход Q	Выход $\bar{Q}$	Режим триггера
1	0	1	0	Установка 1
0	1	0	1	Установка 0
0	0	Последние значения		Хранение информации
1	1	**ЗАПРЕЩЕНО!**		

Схема для эксперимента. Схема, которую необходимо собрать для проведения эксперимента, приведена на рис. 4.9. Она взята из статьи в старом номере журнала "Радио" [6], которая была опубликована в рубрике "Практикум для начинающих".

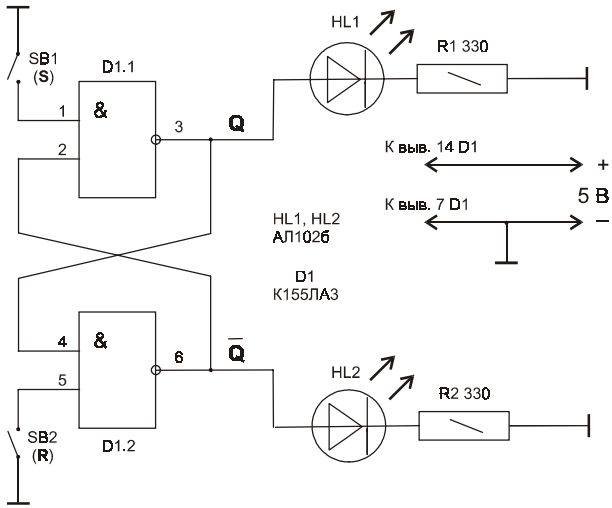


Рис. 4.9. Схема для изучения работы триггера

Для ее сборки необходимо найти некогда очень распространенную микросхему К155ЛА3, содержащую в своем составе 4 логических элемента 2И-НЕ (на рис. 4.8, *а* использованы другие логические элементы, но логика работы экспериментальной схемы практически такая же). Такие микросхемы почти наверняка есть на любой плате от старых вычислительных устройств. Для индикации желательно иметь два любых светодиода с ограничивающими их ток сопротивлениями. Когда в точке, к которой подсоединен диод, высокое напряжение (что соответствует логической 1), он должен светиться. Опыт вполне можно произвести и с одним светодиодом, подключая его к выходам триггера по очереди. В крайнем случае, можно обойтись вообще без светодиодных индикаторов, заменив их любым вольтметром с пределом измерения 5—10 В. Кнопки **SB1** и **SB2**, показанные на схеме, также не являются обязательными — вполне можно просто замыкать проводники. И еще необходим любой источник питания на 5 В, при отсутствии которого можно воспользоваться тремя обычными батарейками на 1,5 В, соединив их последовательно. Таким образом, при минимальной комфортности опыта потребуется всего несколько деталей.

Рассмотрим схему более подробно. Цифры возле выводов логических элементов микросхем обозначают номера выводов. Для того чтобы найти нужный вывод на реальной микросхеме, воспользуйтесь рис. 4.10, где показано,

как принято стандартно нумеровать выводы микросхем данной серии [74]. Не забудьте на выводы **7** и **14** подвести питание, причем ни в коем случае не перепутайте полярность питающего напряжения. Возможно, некоторым стоит напомнить, что все концы проводников, заканчивающиеся перпендикулярной линией (см. нижний по схеме вывод кнопки **SB2**), надо подсоединять к общему проводу — радиолюбители условно называют его "землей".

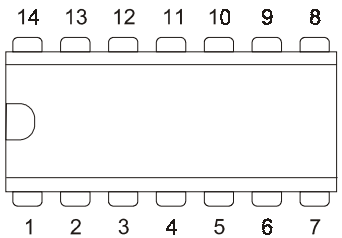


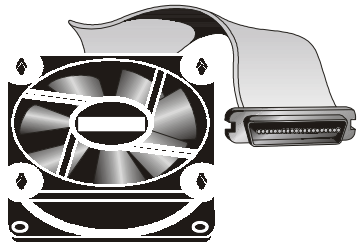
Рис. 4.10. Расположение выводов микросхемы

После сборки внимательно проверьте произведенный монтаж, и только после этого подайте на схему напряжение питания. Сразу же должен загореться один из светодиодов. При замыкании кнопки на входе **S** зажигается светодиод **HL1** (триггер переходит в единичное состояние), а при нажатии другой кнопки — гаснет. Второй светодиод ведет себя противоположным образом.

Внимательно сравните поведение триггера с табл. 4.2, считая нулем разомкнутое состояние кнопок, а единицей — замкнутое. Обязательно убедитесь в том, что при отсутствии входных сигналов триггер сохраняет свое состояние.

Вывод. Триггер способен хранить 1 бит информации и действительно работает в соответствии с табл. 4.2.

ЛЕКЦИЯ 5



Память ЭВМ: устройства внешней памяти

Данная лекция фактически является прямым продолжением предыдущей. Поэтому если вы читаете ее после некоторого перерыва, мой вам убедительный совет просмотреть еще раз *лекцию 4*, особенно ее начало. Хочется особо подчеркнуть, что именно в *лекции 5* будут обсуждаться вопросы взаимодействия всех видов внутренней и внешней памяти, так что с этой точки зрения повторение также будет полезно.

Итак, мы уже много знаем о внутренней памяти ЭВМ, которая позволяет машине производить решение поставленной задачи. Но, тем не менее, остается еще одна часто возникающая на практике проблема — сохранение программы и (или) данных для продолжения работы в будущем. Для этой цели и была создана внешняя память, с которой мы сейчас познакомимся (строго говоря, в современных компьютерах названная цель не является единственной, *см. разд. 5.3.2*).

5.1. Назначение и виды внешней памяти

Классическая функция внешней памяти состоит в том, чтобы ***сохранять информацию*** (как программу, так и данные всех видов) ***для повторного использования***. Отсюда следует, что всю информацию целесообразно хранить во внутреннем двоичном представлении, т. к. это позволяет в случае необходимости немедленно обрабатывать ее без всяких дополнительных преобразований. Именно названное обстоятельство позволяет легко отличать внешнюю память от устройств вывода (или ввода), которые, напротив, используют выходные (входные) данные в виде, удобном для восприятия человека. В частности, любые дисковые устройства по указанной выше причине являются внешней памятью, а сканер или принтер относят к устройствам ввода/вывода. В то же время, все эти устройства можно считать периферийны-

ми или внешними, учитывая одинаковый способ их взаимодействия с центральным процессором (вспомните функциональную схему рис. 2.6).

Как и раньше, мы будем рассматривать разнообразные устройства внешней памяти в историческом плане, следуя той последовательности, в которой они появились. Полезность подобного подхода изучения мы уже обсуждали ранее в *разд. 1.2.*

5.1.1. Внешняя память на бумажных носителях

Сначала устройства внешней памяти использовали в качестве материала для носителя информации обыкновенную бумагу. Вы уже, наверное, догадались, что речь идет о широко распространенных в те годы перфолентах и перфокартах, на которых единица кодировалась пробивкой отверстия, а ноль — его отсутствием. О работе с перфолентой в машинах второго поколения мы уже немного говорили в *разд. 2.2.2.* А вот перфокарты с кодированием информации по строкам и по колонкам изображены на рис. 5.1.

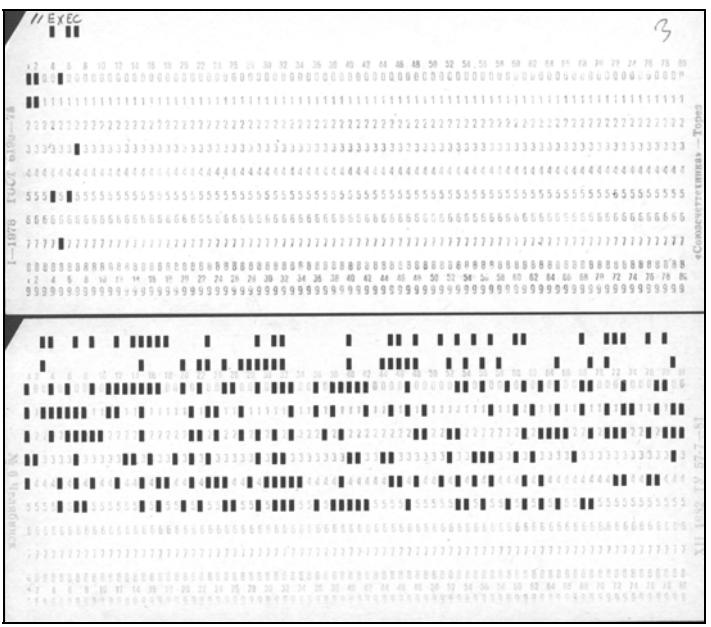


Рис. 5.1. Перфокарты

Возможно, сейчас уже трудно представить, но именно из таких носителей еще совсем недавно, на ЭВМ третьего поколения, составлялись "колоды" программ. Обычно каждому законченному действию соответствовала отдельная карта, что позволяло программисту легко и оперативно вносить изменения.

Примечание

Справедливости ради следует отметить некоторую специфику перфокарт: в основном их применяли для подготовки информации, а не для хранения результатов ее обработки. Иными словами, там чаще всего хранился именно исходный текст программы, а не результат ее трансляции в машинные коды. Такую особенность можно объяснить, вспомнив, что ЭВМ третьего поколения были коллективными и в связи с этим данные для них готовились заранее и на отдельных устройствах. В персональных компьютерах необходимость в такой предварительной подготовке исчезла.

Для дальнейшего сопоставления следует подчеркнуть два достаточно очевидных факта. Во-первых, каждому понятно, что информация на бумажных носителях не могла быть повторно перезаписана (исключая, конечно, имевшие место на практике "трюки" с прорезанием или заклеиванием отверстий). А во-вторых, весь поиск в архивах информации производил оператор ЭВМ "вручную". Для этого перфоленты и перфокарты обычно подписывались, например, как на верхней карте рисунка: слева ручкой нанесен содержащийся на карте текст (попутно отметим, что "exec" — это оператор языка управления заданиями на ЕС ЭВМ, который запускал программу), а справа торопливым почерком указан номер перфокарты — 3.

5.1.2. Внешняя память на магнитных носителях

На смену описанным в предыдущем разделе бумажным носителям пришли магнитные. Вместо отверстий, распознаваемых оптическим способом, на них единица кодировалась наличием намагниченного "пятнышка", а его отсутствие означало ноль.

Первые магнитные устройства по идеям построения в основном повторяли бумажные ленты и перфокарты — просто их заменили магнитными лентами и картами. Информационные емкости устройств благодаря такой замене существенно возросли и, что немаловажно, носители стали многоразовыми.

Появились и новые черты. Отметим, в частности, что магнитная лента для облегчения поиска информации имела специальную служебную разметку. И хотя процесс "каталогизации" по-прежнему велся человеком на бумаге, первые предпосылки для автоматического учета информации на носителях были созданы.

Существенным недостатком магнитных лент являлся последовательный характер доступа к информации (см. рис. 5.2, *а*): чтобы записать данные в конец бобины с лентой, необходимо было ее перемотать механически. И хотя инженеры разработали весьма сложную механику для лентопротяжных механизмов, включавшую специальные форсированные режимы перемотки ленты, все равно хранить большие связанные объемы информации на магнитной ленте было неудобно. Требовалось придумать какие-то новые способы доступа к данным на магнитном носителе.

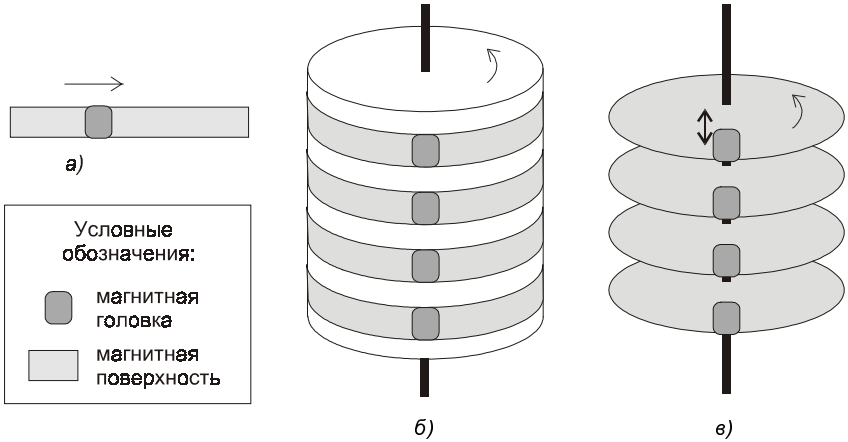


Рис. 5.2. Эволюция устройств магнитной записи: лента, барабан, диск

Весьма естественным шагом в этом направлении явилась разработка магнитных барабанов, на которых лента была кольцевой, а магнитная головка не единственной. Доступ к данным в таком устройстве осуществлялся за счет вращения барабана (рис. 5.2, *б*) и переключения головок, что существенно ускоряло нахождение требуемой информации по сравнению с обычной лентой.

Наконец, последними усовершенствованиями, которые привели к появлению столь привычных нам сейчас магнитных дисков, было изменение формы носителя и, что гораздо важнее, обеспечение движения магнитной головки поперек диска (рис. 5.2, *в*). Дальнейшая эволюция устройств магнитной записи в основном обеспечивалась улучшением качества магнитного покрытия и повышение плотности записи информации.

Особо подчеркнем, что накопители информации на магнитных дисках помимо собственно данных обычно содержат специальные каталоги, с помощью которых компьютер может найти требуемую информацию. Ведением каталогов занимается главная служебная программа — операционная система.

Улучшение качества магнитных дисков, увеличение их емкости и скорости доступа к данным практически вытеснили из употребления магнитную ленту, которая теперь почти не применяется даже для целей резервного копирования больших объемов информации.

Все вы, конечно, не раз пользовались магнитными дискетами, которые по сложившейся традиции различают по измеренному в дюймах диаметру. В настоящее время типовые носители имеют размер 3,5 дюйма; более старые форматы составляли 5,25 и даже 8 дюймов (см. рис. 5.3).

Рис. 5.3. Дискеты 8, 5,25 и 3,5 дюйма

Изготовленные по самым современным технологиям последние модели магнитных устройств внешней памяти, которые при записи к тому же дополнительно используют сжатие информации, позволяют существенно увеличить хранимый на дискете объем. Так емкость ZIP-диска производства фирмы Iomega (рис. 5.4) достигает 250 Мбайт, а JAZ — 2 Гбайт, что может заменить сотни и даже тысячи обычных дискет.

Рис. 5.4. ZIP-дискета емкостью 100 Мбайт и устройство для работы с ней; сейчас емкость таких дисков в несколько раз больше

Очень большую роль в повседневном функционировании компьютера играет так называемый *жесткий диск*, часто на компьютерном жаргоне именуемый *винчестером*[1]. Он расположен внутри системного блока и не виден, пока не снята крышка блока. На жестком диске хранится все основное программное обеспечение и именно с него в подавляющем большинстве случаев происходит загрузка компьютера. Этот процесс мы подробнее рассмотрим в *лекции 7*.

Пролистав при подготовке данной лекции стоящие на полке учебники, я с удивлением обнаружил, что почти в каждом из них имеется рисунок или фотография жесткого диска со снятой крышкой, но почти нигде толком не описывается, как все это впечатляющее хозяйство работает. А сколько там интересного!

Каждому лектору, конечно, хочется, чтобы его курс был не хуже других; поэтому на рис. 5.5 автор тоже приводит фотографию (неисправного!) жесткого диска, в корпусе которого проделаны специальные вырезы, открывающие обзор его внутренних узлов. Вырез, расположенный в верхней части фотографии, позволяет отчетливо увидеть высококачественную зеркальную поверхность диска и головки для считывания информации. Другой, менее заметный вырез в правом нижнем углу дает возможность разглядеть плату управления механизмом.

Рис. 5.5. Как устроен винчестер

[1] Рассказывают, что один из первых жестких дисков имел два накопителя, значения емкости которых формально совпадали с калибром известного двуствольного охотничьего ружья с таким названием (подробнее см. [39]).

Очень важно отметить, что головки чтения/записи винчестера при работе не касаются магнитной поверхности. Благодаря большой частоте вращения диска, равной 5—15 тысячам оборотов в минуту, возникающий воздушный поток создает подъемную силу, и головки в полном смысле слова "парят" над поверхностью диска, не задевая его. С практической точки зрения указанное инженерное решение — наличие микроскопического зазора между головкой и магнитной поверхностью, позволяет резко уменьшить износ магнитного покрытия диска и самих головок. Весь механизм сбалансирован настолько точно, что играет роль даже плотность воздуха: как указано в [И-9], на высоте 2—3 км ее будет недостаточно для того, чтобы оторвать головку от поверхности.

Вся высокоточная механика чтения/записи помещена в корпус, обычно называемый *гермоблоком*. Вопреки распространенным представлениям, он не является герметичным и сообщается с окружающим воздухом через специальный фильтр (при полной герметичности перепад давлений, например, при перевозке в грузовом отсеке самолета мог бы привести к деформации и порче прецизионного механизма). Задача упомянутого фильтра — задерживать твердые частицы, находящиеся в воздухе и не допускать их попадания внутрь гермоблока. Другой фильтр располагается внутри корпуса так, чтобы улавливать отлетающие от поверхности диска частицы.

В отличие от дискет и старых винчестеров, разметка низкого уровня на современных жестких дисках всегда выполняется на заводе-изготовителе, причем для этого используется специальный высокоточный технологический стенд. Иначе говоря, потребитель получает уже размеченный диск с привычными дорожками и секторами. Систему служебных меток часто называют сервоинформацией. Ее роль в работе накопителя исключительно велика: фиксируя прохождение меток под головкой, винчестер стабилизирует частоту оборотов, находит нужные дорожки и правильно располагает информацию при записи секторов.

Электронная плата, управляющая работой диска, также достаточно сложна. Современный винчестер фактически представляет собой самостоятельную ЭВМ с собственным процессором, памятью и даже ПЗУ. Благодаря этому производители накопителей получают возможность применять всевозможные внутренние усовершенствования, улучшающие характеристики своих изделий, не нарушая совместимости с остальными блоками компьютера.

Как видно из сказанного выше, жесткий диск компьютера является очень сложным прибором и требует аккуратного обращения. Прежде всего, следует тщательно оберегать его от ударов и толчков, которые могут повредить точнейшую механику накопителя. Осторожность необходима и при переноске отключенного винчестера, поэтому использование его в качестве "большой дискеты", постоянно носимой в кармане, настоятельно не рекомендуется.

Примечание

Для уменьшения вероятности повреждений магнитной поверхности жестких дисков в нерабочем состоянии, при исчезновении напряжения питания головки автоматически отводятся в специальную *парковочную зону* (на очень старых винчестерах это делалось путем запуска перед выключением питания специальной служебной программы). Разумеется, наличие автопарковки ни в коем случае не отменяет необходимости оберегать выключенный накопитель от ударных воздействий.

По некоторым данным [И-1, И-4] требуется и еще одна предосторожность — речь идет о нежелательности курения за компьютером. Частицы, в огромном количестве присутствующие в табачном дыме, могут проникать через защитный фильтр, что приводит к постепенному нарушению процесса считывания информации (разумеется, здесь речь идет только о сохранности накопителя на жестких магнитных дисках; что касается здоровья пользователей, то существуют гораздо более серьезные причины воздерживаться от курения, причем не только при работе на компьютере).

Таковы некоторые наиболее интересные детали работы современных дисковых накопителей. Имеются очень хорошие и доступно написанные материалы по этому вопросу. Прежде всего, не могу не упомянуть серию тематических публикаций в педагогической газете "Информатика" [36—40], вышедших в традиционной рубрике "На стенд в кабинет информатики". Убедительно рекомендую также прочесть замечательную статью Евгения Музыченко [И-8], доступную в Интернете. Еще один заслуживающий, на мой взгляд, внимания адрес [И-9], где вы найдете высококачественные цветные фотографии "внутреннего строения" винчестера и почерпнете много интересных сведений из опыта ремонта этих устройств. Наконец, наиболее любознательные читатели могут обратиться к обстоятельной книге Михаила Гука [16].

5.1.3. Внешняя память на оптических носителях

Для хранения больших объемов информации, неизбежно возникающих при компьютерной обработке графических изображений и анимации, а также звука и видео, емкости даже самых лучших магнитных дискет часто не хватает, несмотря на все более совершенствующиеся методы сжатия информации. В связи с этим появились новые способы записи данных с использованием оптических носителей. Часто их называют лазерными дисками, поскольку лазер используется как при записи, так и при чтении информации.

Первые оптические диски, получившие названия *компакт-дисков* или сокращенно *CD* (Compact Disc), были созданы компаниями Sony и Philips и предназначались для записи музыки. Достоинства новых носителей были очень быстро оценены, и их начали использовать для хранения компьютерной информации. Дискам с компьютерными данными несколько неудачно, как впоследствии оказалось, присвоили имя CD-ROM. Как вы, вероятно,

помните, ROM расшифровывается Read Only Memory — память ***только*** для чтения, а через некоторое время, вопреки названию, появилась возможность ***записи*** на CD непосредственно на компьютере, и даже удалось разработать компакт-диски с возможностью перезаписи. Таким образом, вместо устройства, хранящего неизменную информацию, оптический диск стал фактически некоторой разновидностью дискеты очень большой емкости.

Один из возможных способов нанесения информации на компакт-диск состоит в том, что луч лазера "выжигает" на нем спиральную дорожку (как на старых грампластинках, только у CD, в отличие от последних, дорожка начинается с внутренней кромки диска и заканчивается на внешней). Мощность луча зависит от сохраняемых данных, так что в результате записи на диске образуется рельеф из более глубоких участков и выступов, который определенным образом кодирует нули и единицы. При считывании также применяется луч лазера, только меньшей интенсивности, чтобы не разрушить записанную информацию. Для распознавания нулей и единиц используется фиксация фактов изменения отражения от поверхности диска в момент перепада высоты информационной спирали. Разумеется, в технических деталях этот процесс выглядит гораздо сложнее. Достаточно сказать, что лазерная головка должна непрерывно отслеживать раскручивающуюся спираль с данными, шаг которой составляет всего 1,6 мкм [68], и суметь найти ее продолжение в случае небольших повреждений. Но мы с вами договорились в рамках нашего курса не обсуждать технические детали, которые, возможно, в скором времени будут изменены в связи с разработкой новых технологий.

Объем информации, которая может быть сейчас сохранена на компакт-диске, составляет 650—700 Мбайт. Совершенно очевидно, что быстро развивающиеся мультимедийные компьютерные приложения не удовлетворятся такой цифрой и будут стимулировать ее рост.

В заключение упомянем о существовании комбинированных магнитооптических накопителей. В основе их работы лежит изменение оптических свойств поверхности носителя (а именно, направления поляризации отражаемого света) под воздействием магнитного поля при нагреве до высокой температуры мощным лазером. Иными словами, лазер только создает условия для записи, а сама информация формируется изменением магнитного поля. По окончании процесса записи вещество, из которого состоит носитель, охлаждается и перестает реагировать на изменения магнитного поля (отсюда высокая надежность хранения информации в описываемых накопителях). При считывании информации лазерный луч по-разному отражается от участков полученного слоя с разной для нулей и единиц поляризацией, что и позволяет их различить. Несмотря на то, что описанные устройства были созданы достаточно давно и имеют определенные достоинства, они не получили массового распространения из-за малой скорости записи и высокой стоимости.

5.2. Организация данных во внешней памяти

Из предыдущего раздела мы видели, что основными носителями информации для внешней памяти служат магнитные и оптические диски. В данном разделе мы ограничимся рассмотрением только дисков первого типа. Появившиеся позднее оптические диски ради обеспечения совместимости с магнитными устройствами во многом имитируют структуру последних, например, тоже имеют сектора. Некоторые детали устройства CD-дисков будут рассказаны в связи с проведением одного из экспериментов *(см. разд. 5.6.3)*.

5.2.1. Размещение информации на носителях

Начнем с того, что обычный магнитный диск имеет две поверхности, пригодные для размещения информации, которые в технической литературе принято называть *сторонами* (side) диска. Учитывая, что в накопителях на жестких дисках на одной оси может быть установлено несколько дисковых пластин, общее количество сторон может быть больше.

Примечание

Первоначально понятие стороны у жестких дисков действительно определялось конструкцией, и общее количество сторон диска равнялось удвоенному количеству магнитных пластин. Однако в современных винчестерах это положение уже не справедливо, и количество сторон часто представляет собой достаточно большой абстрактный числовой параметр, используемый при вычислении истинного расположения информации.

Каждую поверхность обслуживает собственная *магнитная головка* (head). Все головки собраны в единый механический блок и могут двигаться вдоль радиуса дисков, причем движение это является дискретным, т. е. головки занимают относительно диска только строго определенные положения (например, для современного накопителя на гибких магнитных дисках количество стандартных позиций принято равным 80, хотя на практике есть еще несколько дополнительных). Каждое фиксированное положение головки позволяет дисководу за счет вращения диска получить доступ к некоторой замкнутой "полоске" магнитной поверхности, имеющей форму кольца с центром, совпадающим с осью вращения, — ее принято называть *дорожкой* (track) диска. Для дисководов с несколькими магнитными дисками на общей оси совокупность дорожек, обслуживаемых без перемещения головок (проще говоря, расположенных друг под другом), часто называется *цилиндром* (cylinder).

Наконец, каждая дорожка разделена на отдельные *секторы* (sector), как показано на рис. 5.6. Сектор является неделимой порцией информации и может быть прочитан только целиком. Последней координатой информации

на диске служит номер требуемого байта в секторе; выделением отдельного байта из прочитанного сектора занимается уже не сам накопитель, а обслуживающее его программное обеспечение.

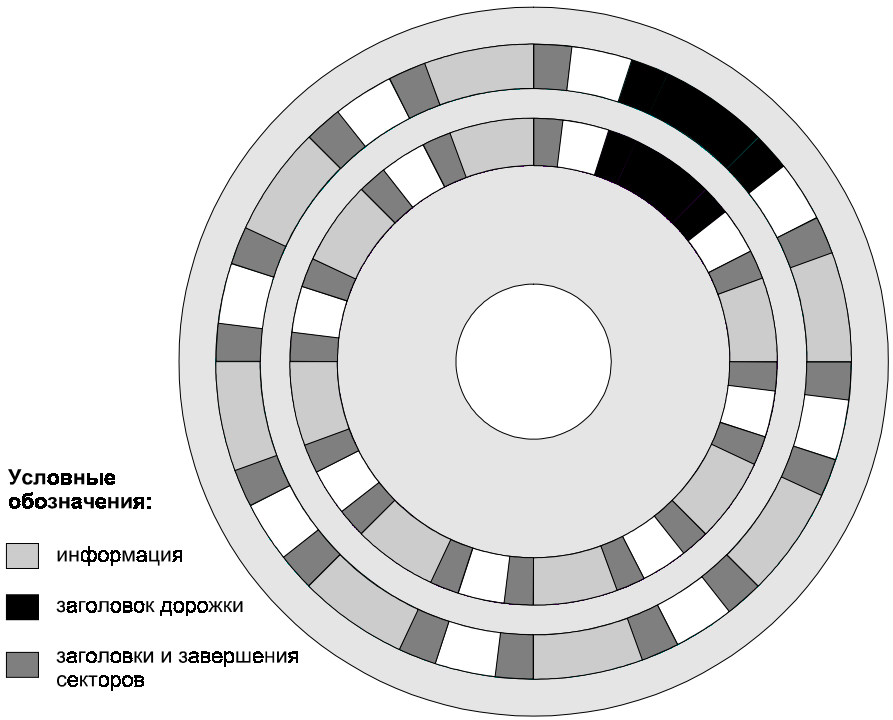


Рис. 5.6. Расположение информации на магнитном диске

Из рис. 5.6 отчетливо видно, что сектора на внешних дорожках диска имеют бóльший размер, чем ближайшие к центру. Это дает основания считать, что данные дорожки должны записываться более надежно. Отсюда нулевая дорожка, обычно используемая для хранения наиболее важной системной информации, всегда размещается на внешнем кольце поверхности диска.

Заметим, что современные жесткие диски могут размещать на разных дорожках разное число секторов, так что для них схема расположения секторов будет более сложной, чем на нашем рисунке. По причине усложнения геометрии расположения информации на диске, написанные на корпусе последних моделей винчестеров параметры, как уже указывалось, являются условными и совсем не обязательно соответствуют фактической конструкции диска.

Из анализируемого рисунка также следует, что помимо собственно информации на диске имеются и служебные данные — своеобразная разметка, облегчающая накопителю поиск нужной информации. Такая разметка создает-

ся в процессе подготовки магнитного носителя к работе и называется *форматированием низкого уровня*. В заголовках записаны характеристики дорожек и секторов (их номера, размер и т. п.), а после каждого сектора помещена контрольная сумма всех его данных: при неправильном считывании сектора сумма получится другой и это позволит зафиксировать ошибку.

Как ни странно, сектора на дорожке совсем не обязаны быть пронумерованы по порядку. Широко известен способ чередования секторов, когда их располагают на дорожке дискеты или винчестера не последовательно, а, например, в порядке 1-4-7-2-5-8-3-6-9. Делается это для того, чтобы компьютер успевал получить все данные из сектора "до подхода" следующего [9]. Для современных моделей, обеспечивающих высокую скорость считывания данных, подобный метод становится не актуальным.

В отличие от описанного физического форматирования на низком уровне, существует еще логическое форматирование. Логический формат диска существенно зависит от того, как операционная система организует на нем информацию. В процессе логического форматирования каждая ОС создает на дискете или на винчестере собственные служебные области, например каталог файлов, и очищает их.

Для практики полезно знать различие между логическим и физическим форматированием. Так, например, в случае возникновения на дискете плохих блоков стоит производить именно полное, а не ускоренное (чисто логическое) форматирование, ибо тогда происходит восстановление испорченной разметки, что часто позволяет избавиться от плохих секторов. Напомним, что форматирование жесткого диска на низком уровне является значительно более сложным процессом и не всегда возможно "в домашних условиях" *(см. разд. 5.1.2)*.

Некоторую дополнительную информацию о процессе форматирования читатели также смогут получить в ходе экспериментов с дискетами, которые описаны в *разд. 5.6.4*.

5.2.2. Доступ к информации на внешних носителях

Итак, положение интересующего нас байта информации на магнитном диске определяется четырьмя "координатами": номером стороны, номером дорожки диска, номером сектора и номером байта в нем. Часто о такой организации доступа говорят как о доступе к диску на физическом уровне. Описанная выше сложная система хранения данных требует определенных усилий по их извлечению со стороны программного обеспечения. Сравните это со схемой доступа ко внутренней памяти, описанной в *лекции 4*, и вы легко поймете, почему все программное обеспечение сначала копирует данные из внешней памяти в ОЗУ и лишь затем их обрабатывает.

Работой с дисками на физическом уровне, т. е. в терминах "сторона-дорожка-сектор", занимаются обычно программы, размещенные в ПЗУ компьюте-

ра. Как мы уже знаем *(см. разд. 4.1.2)*, их принято называть BIOS, т. е. Basic Input/Output System. Операционная система хотя и опирается на BIOS, тем не менее, всегда пользуется сквозной логической нумерацией секторов без учета их физического расположения. Для пересчета логического номера сектора в его физические координаты и наоборот существуют специальные формулы. В случае постоянного количества секторов на дорожке они совсем не сложные, в чем можно убедиться, разобрав материалы эксперимента, предложенного в *разд. 5.6.2*.

5.2.3. Файловая система

Рядовой пользователь обычно не имеет представления ни о физических, ни даже о логических номерах секторов, в которых записана его информация. Для обращения к ней он манипулирует определенной системой имен, принятой в данной версии операционной системы. Зная имя файла с информацией, система обращается к размещенным на диске специальным служебным таблицам и по ним определяет номера требуемых логических секторов, а затем вычисляет их физическое местонахождение и производит чтение или запись.

Не все также знают, что сектора с информацией данного файла совсем не обязательно располагаются по порядку в одном месте диска. При записи система активно использует свободные места, которые образуются при удалении ненужных файлов. В результате отдельные части файла вполне могут попасть в разные области диска, что будет заметно замедлять доступ к информации. Для устранения этого явления в состав операционных систем обычно входят специальные служебные программы дефрагментации файлов.

Честно говоря, обсуждая важную роль операционной системы в доступе к дисковой информации, мы несколько забежали вперед — роли программного обеспечения в работе компьютера будет посвящена *лекция 7*. Тем не менее без подобного упоминания описание доступа к диску было бы неполным.

5.2.4. Роль контроллеров

В заключение еще одно важное замечание о работе устройств внешней памяти. Поскольку все они, как и устройства ввода/вывода, подсоединяются к шине данных через контроллеры (снова сошлемся на схему рис. 2.6), процедура обмена во многом определяется именно логикой работы контроллера.

Контроллеры современных дисковых накопителей обладают достаточно высоким интеллектом. Так, выше уже отмечалось, что контроллер винчестера фактически представляет собой самостоятельную ЭВМ и способен реализовывать довольно сложные действия. Например [И-8], он "помнит" все имеющиеся на магнитной поверхности некачественно изготовленные сектора (а их при современной плотности записи избежать не удается!) и спосо-

бен подменять их резервными, создавая видимость диска, который полностью свободен от дефектов. Или еще один пример. Современные жесткие диски используют технологию S.M.A.R.T. (Self Monitoring Analysis and Report Technology — дословно "технология самоконтроля, анализа и отчета"; кроме того, английское слово "smart" имеет значение "разумный", "интеллектуальный"). Винчестер, оснащенный такой возможностью, со времени самого первого включения ведет статистику своих параметров, сохраняя ее результаты в некоторой скрытой области диска. Накопленные данные могут помочь специалистам при анализе состояния жесткого диска и условий его эксплуатации. В эксперименте, описанном в *разд. 5.6.5*, мы познакомимся со S.M.A.R.T.-информацией несколько подробнее.

Еще одним контроллером, обеспечивающим достаточно сложное поведение, является контроллер накопителя на CD-дисках. Вот как выглядит логика его работы при поиске нужного кадра (части сектора); еще раз напомним читателям, что оптический диск не разделен на отдельные дорожки, а представляет собой единую спираль, так что сделать это не так просто. "Для считывания данных контроллер дисковода CD-ROM предпринимает следующие действия:

1. Располагает головку как можно ближе к предполагаемому местоположению искомого кадра.
2. Ждет, пока диск провернется настолько, что углубления и ровные участки окажутся непосредственно под лазерным лучом. Затем начинает перемещать головку вдоль спирали к внешней стороне диска.
3. Ждет, пока головка не совместиться с началом какого-либо кадра, а затем читает его адрес.
4. Исходя из расположения искомого кадра относительно головки, перемещает ее по нужному адресу".

Приведенная цитата заимствована из книги [68].

5.3. Взаимодействие различных видов памяти

Итак, уважаемые читатели, мы познакомились с самыми разнообразными видами как внутренней, так и внешней памяти. Обобщая наши знания, рассмотрим в данном разделе, как части памяти связаны между собой и как они взаимодействуют друг с другом.

5.3.1. Взаимодействие внутренней и внешней памяти

Сравнивая способы доступа к информации во внутренней *(см. разд. 4.2.1)* и внешней *(см. разд. 5.2.2)* памяти, можно видеть, что они реализованы суще-

ственно по-разному. Любой байт команды или данных может быть немедленно извлечен из ОЗУ или ПЗУ по заданному адресу, выставленному на адресную шину. Что же касается памяти внешней, то тут, как мы уже видели, необходимы более значительные усилия. Сначала требуется найти на носителе блок с необходимой информацией (для дисков им является сектор) и считать его целиком в рабочую область памяти, называемую *буфером*. И только потом появится возможность извлечь из буфера требуемый байт информации. Таким образом, прежде чем использовать какие-либо команды программы или данные, хранящиеся на диске, их необходимо предварительно считать во внутреннюю память.

Наличие файловой системы на современных носителях делает процедуру извлечения информации еще более сложной. Обращение к файлу происходит по имени, поэтому операционная система предварительно обращается к системной информации в каталоге и находит там сведения о физическом расположении затребованного файла.

Обсуждая внутреннюю и внешнюю память, авторы используют множество аналогий, в том числе достаточно интересных. Тем не менее, не следует им чрезмерно увлекаться. Вот как пишет об этом Ч. Петцольд [67]:

> "На вопрос о постоянстве и оперативности[1] часто отвечают, используя такую аналогию: "Считайте, что оперативная память — это ваш рабочий стол, а постоянная — шкаф с папками". И думают, что это прекрасное сравнение! Но я его таковым не считаю: создается впечатление, что архитектура компьютера срисована с обычной конторы. Правда же заключается в том, что различие между постоянной и оперативной памятью искусственно и существует лишь потому, что нам до сих пор не удалось создать накопитель данных, который был бы одновременно быстрым, объемным и способным хранить информацию в течение долгого времени".

В качестве дополнения к мысли о некоторой искусственности разделения памяти рассмотрим вопрос о виртуальном способе адресации памяти, принятом в современных компьютерах, при котором внутренняя и внешняя память рассматриваются как единое целое.

5.3.2. Виртуальная память

Пользователям хочется, чтобы программное обеспечение было интеллектуальным и дружественным и чтобы в нем были предусмотрены все самые мелкие детали, которые им (пользователям) могут потребоваться. Программистам хочется написать программу с наименьшими затратами сил и времени, поэтому они широко используют системы программирования все более

[1] Автор цитаты использует термины "постоянная" и "оперативная" память вместо принятых в лекции "внешняя" и "внутренняя".

и более высокого уровня. Обе названные причины приводят к одному и тому же: программы получаются прямо-таки гигантскими по объему. Так, например, размер только одного исполняемого файла Excel более 5 Мбайт, а известный векторный графический редактор CorelDRAW и того больше — 13 Мбайт. К этому добавляется объем обрабатываемых данных, который также постоянно растет. И еще не следует забывать об операционной системе и о многозадачном режиме работы современной вычислительной техники — следовательно, памяти требуется еще больше.

Как же согласовать эти требования с большим, но все же ограниченным объемом ОЗУ? Довольно изящным способом преодоления данной трудности является использование *виртуальной памяти*. Современные операционные системы типа Windows работают в предположении, что компьютер обладает огромным объемом памяти, так что реально установленное ОЗУ — лишь некоторая часть этого пространства. А где же тогда находится оставшаяся часть? Совершенно верно — во внешней памяти или, более точно, в специальном системном файле на жестком диске. Теперь, если ОЗУ по какой-то причине не хватает, система копирует "наименее нужную" в данный момент область ОЗУ на диск, освобождая тем самым необходимый объем памяти. Когда, наоборот, потребуются данные с диска, то они будут возвращены в освобожденное описанным ранее способом место ОЗУ (заметьте, что это совсем не обязательно будет то же самое первоначальное место!)

Наших знаний о способах взаимодействия процессора с различными видами памяти вполне хватит, чтобы сразу же сделать вывод о медленности работы описанного механизма по сравнению с обычной адресацией памяти. В самом деле, если в компьютере установлен недостаточный объем ОЗУ, то программы выполняться будут, но необычайно медленно из-за постоянного обмена данными между ОЗУ и жестким диском. Кстати, такой напряженный режим работы во многих случаях хорошо заметен на слух и по постоянному миганию индикатора обращения к винчестеру. В подобной ситуации необходимо увеличить объем установленного на плате ОЗУ, и компьютер сразу станет работать во много раз быстрее.

5.3.3. Иерархия памяти

Все виды компьютерной памяти связаны между собой, образуя своеобразную иерархию.

Из приведенной схемы можно заметить, что чем "выше" рассматриваемая разновидность памяти, т. е. чем ближе она к процессору, тем меньше ее объем, но зато тем больше скорость ее работы. Иерархия памяти специально спроектирована так, что более нужная для данной задачи информация попадает в память более высокого уровня. Например, файлы, в которых хранятся требуемые для решения задачи данные, считываются в ОЗУ, а наиболее часто используемые из них через какое-то время попадут в кэш-память.

Подчеркнем, что термин "кэширование" в вычислительной технике имеет довольно широкий смысл: сохранение информации в более быстродействующей памяти с целью повторного использования. В таком контексте ОЗУ кэширует внешнюю память, а данные из Интернета сохраняются на винчестере для ускорения загрузки при повторном использовании (речь идет о так называемых временных файлах Интернета, которые хранятся в специальной служебной папке). В дополнение к сказанному заметим, что и внутри внешних устройств компьютера, например, самого накопителя на жестких магнитных дисках, тоже возможно кэширование.

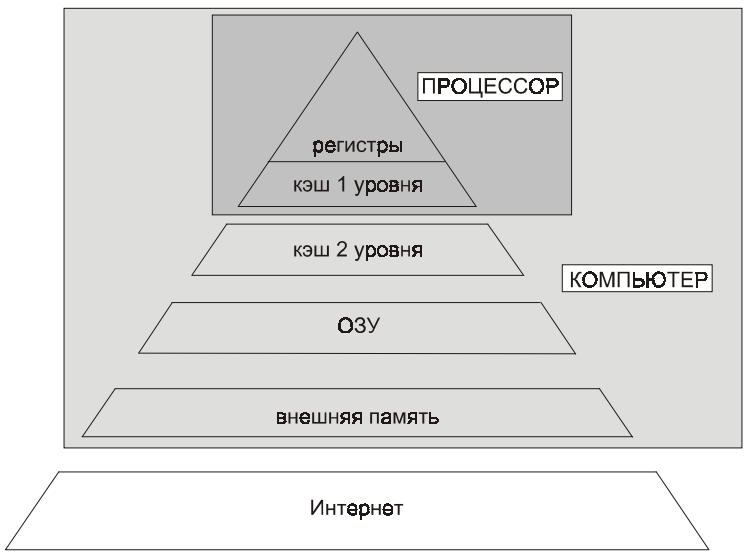


Рис. 5.7. Иерархия видов памяти

Подчеркнем, что для эффективного функционирования всех иерархических уровней памяти они должны быть хорошо согласованы между собой.

Говоря об иерархии памяти, мы пренебрегали техническими деталями обмена информацией между устройствами компьютера (например, ролью контроллеров и центрального процессора в реализации чтения данных). В этом вопросе автор считает подобную абстракцию правомерной, однако не надо ее абсолютизировать и переносить "на все случаи жизни".

5.4. Выводы

После рассмотрения всех видов памяти можно сделать следующие выводы.

- Основная функция внешней памяти ЭВМ состоит в том, чтобы сохранять информацию для повторного использования. В современных компьюте-

рах она также может использоваться для кажущегося (виртуального) увеличения объема ОЗУ.

- Характерным свойством внешней памяти является использование данных во внутреннем представлении ЭВМ. Данные на внешних носителях недоступны человеку без помощи компьютера.
- Распространенные в настоящий момент устройства внешней памяти в основном используют магнитные и оптические носители. Они разнообразны по технической реализации, но логические принципы их работы во многом схожи.
- Информация на носителях внешней памяти хранится в виде отдельных блоков — секторов. Сектор является неделимой единицей информации, т. е. может быть прочитан только целиком. Выделение информации меньшей величины осуществляется программным путем.
- Диски имеют физический и логический формат. Последний существенно зависит от операционной системы, которая обслуживает диск. Для упрощения (разумеется, для пользователя, а не для ЭВМ) доступа к данным в современных ОС имеется специальная файловая система, заменяющая физические координаты информации символическими именами файлов и каталогов.
- В обмене данными между внешней и внутренней памятью существенную роль играют контроллеры. В современных устройствах внешней памяти контроллеры довольно сложны и фактически являются самостоятельными ЭВМ.
- Любая информация, хранящаяся во внешней памяти, для использования предварительно считывается в ОЗУ.
- Деление памяти на внутреннюю и внешнюю несколько условно. Современные операционные системы способны организовывать их использование как для единой виртуальной памяти.
- Все виды компьютерной памяти связаны между собой, образуя общую иерархию.

5.5. Вопросы для осмысления

1. Постарайтесь вспомнить все виды компьютерной памяти, которые вы знаете. Поясните их основные свойства и функции.
2. Какие из названных вами в предыдущем вопросе видов памяти реально есть в вашем компьютере? Знаете ли вы их объем?
3. Какими носителями внешней памяти вы пользовались? Каков их объем и какую примерно его часть вы использовали? Попробуйте выразить объем всех этих устройств в дискетах, считая для удобства вычислений объем одной дискеты равным 1,5 Мбайт.

4. Почему головки винчестера не царапают магнитные поверхности? Как влияют на работу винчестера механические воздействия, например толчки стола, на котором находится компьютер?
5. Известно ли вам количество плохих блоков на вашем винчестере? Не растет ли оно?
6. Исходя из примерной оценки по 1 Мбайт памяти на 1 сек. звучания (подразумевается распространенный формат MP3, применяющий сжатие данных), оцените, сколько "среднестатистических" эстрадных песен сможет вместить один CD-ROM? А весь ваш винчестер?
7. Можно ли считать с диска один отдельно взятый байт? Как его получить?
8. Что такое форматирование диска? Приходилось ли вам хоть раз выполнять данную процедуру? Если да, то с какой целью? Умеете ли вы делать дефрагментацию диска?
9. Опишите с максимально возможным числом подробностей процесс загрузки определенного файла в память компьютера.
10. Как работает виртуальная память? Попробуйте разработать алгоритм записи и считывания байта из виртуальной памяти (указание: не забудьте разделить память на отдельные страницы фиксированной длины. Подумайте, как можно определять наименее нужную страницу при освобождении ОЗУ).
11. Оцените соотношение объемов всех уровней памяти вашего компьютера (см. рис. 5.7).

5.6. Любопытные эксперименты

5.6.1. Логический доступ к сектору дискеты

Как было описано в *разд. 5.2.2*, различают физический и логический доступ к секторам диска. Более низкий уровень имеет физический доступ, когда для нахождения требуемого сектора требуется задать его номер на дорожке, а также номера самой дорожки и стороны, на которой она находится. Более высоким, а значит и более простым уровнем, является логическое обращение к диску, когда физическая структура диска не рассматривается вообще, а сектора диска предполагаются пронумерованными "подряд". В этом случае достаточно задать логический номер сектора, а программное обеспечение пересчитает его в физические "координаты" (сектор, дорожка, сторона).

По понятным причинам легче всего начать с логического доступа к диску, причем ограничившись самой простой операцией считывания сектора с дискеты.

Цель эксперимента. Научиться считывать с дискеты 1 логический сектор.

Для обеспечения логического доступа к диску в операционной системе обычно существуют специальные функции. Мы воспользуемся наиболее простыми и многократно описанными в литературе (см. [61] или любую аналогичную) функциями системы MS-DOS, которые, к счастью, пока продолжают корректно выполняться из среды Windows.

Для проведения эксперимента потребуется программа, которая приведена в листинге 5.1. К сожалению, несмотря на все старания автора, она все же довольно длинная. Причиной является то, что считанный сектор необходимо выводить на экран в удобном для обозрения виде, на что и уходят основные усилия. В частности, процедура `printHalf` выводит на экран половину прочитанного сектора (как нетрудно подсчитать, весь сектор отобразить в 25 строках экрана не получается). Вывод производится в виде шестнадцатеричных кодов и в виде текста, причем коды со значением менее 32, не имеющие стандартного видимого изображения, традиционно заменяются точками. Для организации вывода используются процедуры более низкого уровня — `hexPrint`, печатающая шестнадцатеричное значение байта, и `printChar`, которая выводит один шестнадцатеричный символ. Ниже приведено краткое описание перечисленных процедур вывода.

Процедура `printChar`. Анализирует передаваемое ей в качестве аргумента число и для значений `c` < 10 преобразует его в соответствующую цифру, а для остальных (10—15) производит его пересчет по общепринятым правилам в заглавную латинскую букву 'A' — 'F'.

Процедура `hexPrint` выделяет старшую и младшую цифру байта путем деления на 16 и затем вызывает предыдущую процедуру. После вывода двух цифр на экран дополнительно выводится пробел.

Процедура `printHalf` заметно сложнее. Она призвана организовать весь процесс печати половины буфера `buf`, в который считан интересующий нас сектор. В случае нулевого аргумента, выводятся байты 0—255, а при любом другом значении байты 256—511, т. е. вторая половина сектора. После печати каждого байта проверяется, не закончилась ли строка: при выводе подобного рода информации принято группировать коды по 16, так что после завершения очередной строки остаток от деления номера следующего символа обращается в ноль или, более кратко, `((i+1) MOD 16) = 0`. При выполнении указанного условия выводится все вошедшие в строку 16 байтов в виде символов, причем, как указывалось выше, управляющие символы с кодами меньше 32 заменяются точкой.

Разобравшись во вспомогательных процедурах вывода на экран, обратимся, наконец, к цели нашего опыта — чтению содержимого определенного сектора в массив `buf`. Прежде всего, в переменную `n` вводится десятичный номер интересующего нас сектора. Затем следует небольшая вставка в машинных командах, которая и читает сектор (приятно поражает, насколько легко в Турбо Паскале делаются ассемблерные вставки: причем не менее просто

ассемблерный фрагмент согласуется с переменными программы на Паскале). После того как в определенные регистры микропроцессора помещены все необходимые параметры, команда `INT 25` вызывает специализированную подпрограмму MS-DOS, читающую сектора диска (в нашем случае запрашивается только один) в указанную область памяти. Вот, собственно, и вся программа: все остальные действия берет на себя операционная система.

Примечание

Последняя инструкция процессора (`pop cx`) "сбрасывает" возвращаемую подпрограммой чтения информацию о возможных ошибках, которую наша программа для простоты игнорирует.

Последняя строка программы выводит содержимое первой половины прочитанного сектора на экран и после нажатия клавиши <Enter> — вторую.

Листинг 5.1. Программа чтения логического сектора дискеты

```
PROGRAM read_sector;
VAR buf:ARRAY [0..511] OF CHAR;
    n,i,k,m:INTEGER;
PROCEDURE printChar(c:byte); {вывод hex-цифры}
BEGIN IF c<=9 THEN WRITE(CHR(c+ORD('0')))
              ELSE WRITE(CHR(c-10+ORD('A')))
END;
PROCEDURE hexPrint(b:byte); {вывод hex-числа и ' '}
BEGIN printChar(b DIV 16); printChar(b MOD 16);
      WRITE(' ')
END;
PROCEDURE printHalf(n:integer);
VAR i1,i2:INTEGER;
BEGIN IF n=0 THEN i1:=0 ELSE i1:=256; i2:=i1+255;
      FOR i:=i1 TO i2 DO
          BEGIN hexPrint(ORD(buf[i]));
                IF ((i+1) MOD 16) = 0 THEN
                   BEGIN m:=(i DIV 16)*16;
                         FOR k:=m TO m+15 DO
                             IF ORD(buf[k])>=32
                                THEN WRITE(buf[k])
                                ELSE WRITE('.');
                         WRITELN
                   END
          END
END;
BEGIN WRITE('Введите десятичный номер сектора: ');
      READLN(n);
```

```
      ASM lea bx,buf {адрес буфера}
          mov dx,n   {номер сектора}
          mov cx,1   {количество секторов}
          mov al,0   {диск A:}
          int 25h
          pop cx
      END;
      printHalf(0); READLN; printHalf(1)
END.
```

Для большего удобства можно путем небольших исправлений переделать нашу программу для работы под Windows. Для этого ее надо откомпилировать в Borland Pascal for Windows, который используется значительно реже. По этой причине мы не будем обсуждать здесь все усовершенствования, ограничившись упоминанием самого важного: благодаря использованию BPW, мы можем получить окно с произвольным количеством строк. Поэтому весь сектор удается отобразить сразу.

Программа sect_win (рис. 5.8), как и программа, приведенная в листинге 5.1, содержится на прилагаемом к книге CD-диске. Причем приведен не только их текст на языке Паскаль, но и результат компиляции, так что сразу можно использовать готовую программу.

Примечание

В отличие от всех предыдущих программ, в sect_win все русские тексты хранятся в кодировке Windows, что необходимо учитывать при ее просмотре.

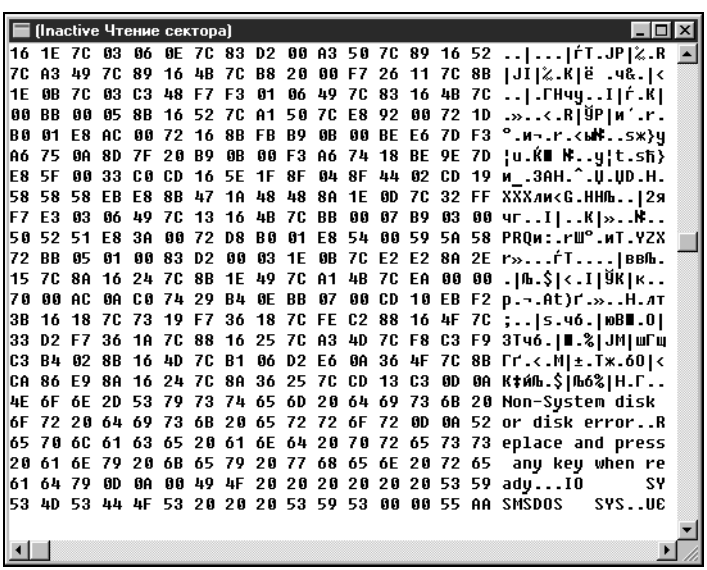


Рис. 5.8. Результат работы программы sect_win

Итак, инструмент для изучения секторов дискеты готов. Мы его еще используем в некоторых следующих опытах, а пока попробуйте почитать некоторые сектора дискеты. Определенный интерес представляет загрузочный сектор 0, а также сектора каталога, начинающиеся с номера 19. А собственно информация файлов (и вложенных подкаталогов) находится в секторах, расположенных начиная с сектора 33.

Если читателей заинтересовала проблема работы с секторами диска, советую обязательно посмотреть книгу [62], в которой обучение программированию ведется на примере разработки утилиты работы с диском.

Вывод. Оказывается, прочитать сектор дискеты совсем несложно. Во всяком случае, если опираться на мощь операционной системы, это гораздо проще, чем вывести прочитанное содержимое на экран в аккуратном систематизированном виде. В качестве вывода по методике экспериментов можно сказать, что даже такое несложное программное обеспечение, как наше, позволяет довольно много узнать о хранении информации на диске. Кроме того, существуют специализированные утилиты для контроля информации на дисках, которые существенно повышают возможности анализа. Одна из таких программ будет использована в экспериментах, описанных в *разд. 5.6.3*.

5.6.2.* Физический доступ к сектору дискеты

Перейдем теперь к физическому доступу к диску. Для изучения этого более сложного вопроса поставим новые эксперименты.

Цель эксперимента. Научиться считывать с дискеты 1 физический сектор.

Поставленная задача фактически состоит из двух частей: вычисление физических параметров расположения сектора и собственно считывание.

Для решения первой проблемы существуют несложные формулы, которые легко выводятся из интуитивных соображений. Рассмотрим для примера, как вычислить номер сектора на дорожке. Пусть на каждую дорожку входит одинаковое количество секторов — для дискеты это предположение соблюдается и соответствующая константа равна 18. Тогда для нахождения искомой величины достаточно вычислить, сколько секторов остается после вычитания содержимого всех полных предшествующих дорожек. Для этой цели в языке Паскаль даже существует специальная операция `MOD` — остаток от деления, которая сразу определяет требуемый нам результат.

Важно

Все вычисления с применением операций `DIV` и `MOD` устроены так, что нумерация объектов должна начинаться с нуля. Для дорожек и сторон диска используется именно такой способ, а для секторов по сложившейся традиции номера начинаются с единицы. Отсюда в соответствующих формулах обычно добавляется или вычитается 1.

Аналогичным образом вычисляются и остальные физические параметры для данного логического сектора. Необходимо лишь учитывать, что для оптимизации доступа к диску секторами сначала заполняются все стороны данной дорожки, только затем происходит переход к следующей дорожке. Например, самые первые 18 секторов располагаются на нулевой стороне первой дорожки, следующие 18 — на первой стороне этой же дорожки, а следующая партия секторов уже попадает на нулевую сторону новой дорожки.

Программа для пересчета номера логического сектора (dos_sector) в физические параметры для функций BIOS (bios_sector, bios_track и bios_side) и в целях контроля обратно приводится на листинге 5.2. Формулы и обозначения, как видно из комментариев в программе, взяты из книги Питера Нортона [61].

Листинг 5.2. Программа пересчета номера логического сектора

```
PROGRAM BIOS_DOS_sector;
{Формулы: П. Нортон, "ПК фирмы IBM и операционная система MS-DOS"
М.: Радио и связь, 1992. См. стр. 237}
CONST sectors_per_side=18; {секторов на дорожке}
      sides_per_disk=2;    {сторон на диске}
VAR dos_sector, n,
    bios_sector, bios_track, bios_side :INTEGER;
BEGIN WRITE('Введите десятичный номер сектора в DOS: ');
      READLN(dos_sector);
{DOS ==> BIOS}
      bios_sector:=1+dos_sector MOD sectors_per_side;
      bios_track :=dos_sector DIV (sectors_per_side*sides_per_disk);
      bios_side  :=(dos_sector DIV sectors_per_side) MOD sides_per_disk;
      WRITELN('BIOS sector: ', bios_sector);
      WRITELN('BIOS track:  ', bios_track);
      WRITELN('BIOS side :  ', bios_side);
{BIOS ==> DOS}
      n:=(bios_sector-1)+bios_side*sectors_per_side
         +bios_track*sectors_per_side*sides_per_disk;
      WRITELN('DOS sector:  ', n);
END.
```

Приведенная выше программа позволяет пересчитать номер логического сектора в номера сектора, дорожки и стороны, которые требуется задать функции BIOS для доступа к сектору на физическом уровне. Теперь, имея в своем распоряжении эти координаты, мы можем перейти к решению второй части задачи — чтения сектора. Программа, предназначенная для этой цели, приведена в листинге 5.3.

Листинг 5.3. Программа чтения физического сектора дискеты

```
PROGRAM read_BIOS_sector;
VAR buf:ARRAY [0..511] OF CHAR;
    sector,track,side:BYTE;
    i,k,m:INTEGER;
PROCEDURE printChar(c:byte); {вывод hex-цифры}
BEGIN IF c<=9 THEN WRITE(CHR(c+ORD('0')))
              ELSE WRITE(CHR(c-10+ORD('A')))
END;
PROCEDURE hexPrint(b:byte); {вывод hex-числа и ' '}
BEGIN printChar(b DIV 16); printChar(b MOD 16);
      WRITE(' ')
END;
PROCEDURE printHalf(n:integer);
VAR i1,i2:INTEGER;
BEGIN IF n=0 THEN i1:=0 ELSE i1:=256; i2:=i1+255;
      FOR i:=i1 TO i2 DO
          BEGIN hexPrint(ORD(buf[i]));
                IF ((i+1) MOD 16) = 0 THEN
                   BEGIN m:=(i DIV 16)*16;
                         FOR k:=m TO m+15 DO
                             IF ORD(buf[k])>=32
                                THEN WRITE(buf[k])
                                ELSE WRITE('.');
                         WRITELN
                   END
          END
END;
BEGIN WRITELN('Введите десятичные номера:');
      WRITE('сектора: '); READLN(sector);
      WRITE('дорожки: '); READLN(track);
      WRITE('стороны: '); READLN(side);
      ASM lea bx,buf    {адрес буфера}
          mov cl,sector {сектор}
          mov ch,track  {дорожка}
          mov dh,side   {сторона}
          mov dl,0      {диск A:}
          mov al,1      {количество секторов}
          mov ah,2      {читать сектора}
          int 13h
      END;
      printHalf(0); READLN; printHalf(1)
END.
```

Сравнив листинги 5.1 и 5.3, обнаружим, что их начальная часть, посвященная выводу содержимого на экран, совпадает. Что касается логики оставшейся части листинга 5.3, то она достаточно прозрачна: вводятся координаты интересующего нас сектора, эти и некоторые другие дополнительные параметры заносятся в определенные регистры микропроцессора и происходит обращение к функции BIOS с номером 2 при помощи инструкции `INT 13`. После этого остается только вывести на экран считанную в массив `buf` информацию.

Примечание

Функция BIOS `INT 13`, как и функция DOS `INT 25`, способна прочитать несколько последовательных секторов. Но функция BIOS по сравнению с функцией DOS менее универсальна и годится только для секторов в пределах одной физической дорожки; функция DOS лишена подобных ограничений.

Набрав и отладив все три приведенные выше программы (или воспользовавшись прилагаемым к книге диском), можно приступить непосредственно к проведению эксперимента. Его можно провести по следующему сценарию. С помощью программы из листинга 5.1 выбираем произвольный сектор дискеты, содержащий некоторый характерный текст. Он позднее поможет нам идентифицировать сектор и тем самым подтвердить правильность результатов эксперимента. Затем выбранный логический номер вводим в программу из листинга 5.2 и фиксируем результаты ее вычислений для дальнейшего использования. Наконец, на завершающем этапе вводим полученные значения в программу из листинга 5.3 и убеждаемся, что она читает и выводит на экран тот самый сектор, который мы ожидали увидеть.

Вывод. Таким образом, указанное совпадение результатов говорит о правильности нашего понимания закономерностей доступа к информации на физическом уровне. Отчетливо видно, что такой доступ к информации сложнее, чем на логическом уровне, и поэтому он используется заметно реже — только в некоторых системных программах обслуживания дисков.

5.6.3. Чтение секторов CD

Освоив чтение секторов с дискеты, мы можем попробовать проделать то же самое с компакт-диском. На этот раз не будем утомлять читателя написанием программы и воспользуемся для проведения эксперимента готовой дисковой утилитой [И-29].

Цель эксперимента. Посмотреть содержимое секторов CD.

В лекции описывались главным образом принципы хранения информации на магнитных дисках. Оптические CD обладают некоторыми особенностями, которые имеет смысл рассмотреть, прежде чем начинать экспериментировать. Мы уже знаем *(см. разд. 5.1.3)*, что на компакт-дисках не существует

дорожек, поскольку данные расположены вдоль одной непрерывной спирали. Пока не используется и еще одна координата — сторона диска, т. к. современный принцип записи информации предполагает всего одну рабочую поверхность. Зато сектора на оптических дисках тоже имеются, хотя принцип их размещения немного сложнее, чем на магнитных дисках. Наименьшая единица информации на CD — это *кадр* (frame) размером 24 байта; кадры группируются в блоки по 98 кадров (2353 байта). На самом деле кадр компакт-диска содержит только 2048 байтов данных, остальные, что вас, видимо, уже не удивит, являются служебными и предназначены для коррекции ошибок, синхронизации чтения и адресации секторов [68].

Вооруженные этими знаниями, запустим названную выше программу-утилиту и выберем в качестве рабочего диска CD-ROM. Попробуем прочитать несколько секторов. Заглянув по привычке в 19-й сектор, мы с удивлением обнаружим там, как и на дискете, каталог (рис. 5.9).

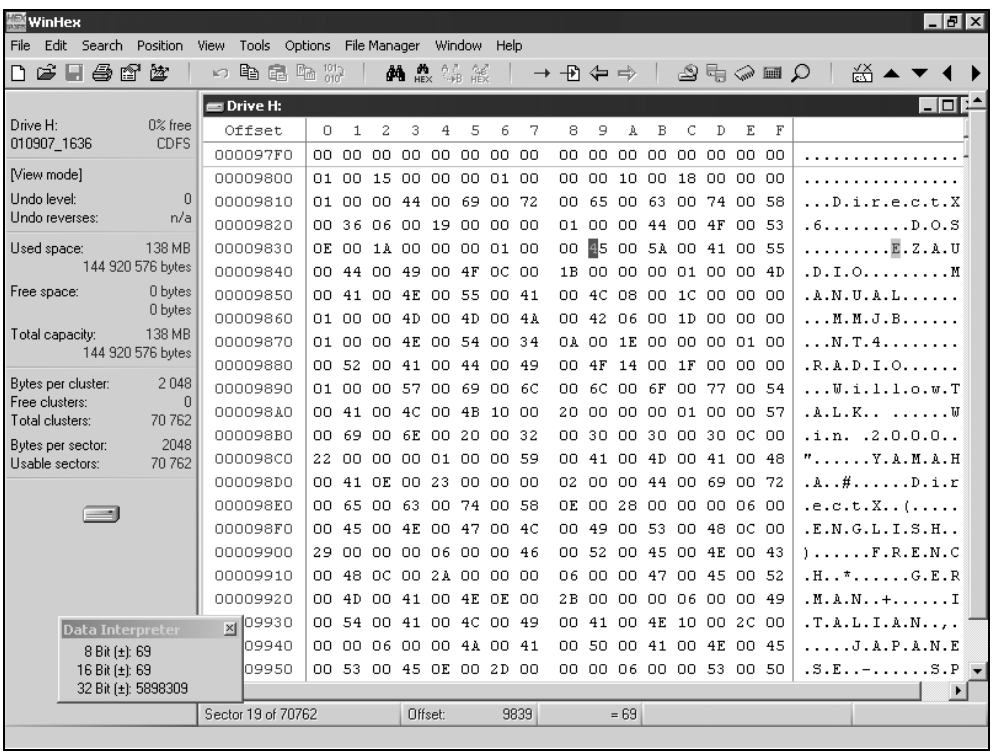


Рис. 5.9. Сектор оптического диска, содержащий каталог

Сопоставим его с рис. 5.10, на котором показано содержимое каталога диска, отображаемое в окне Windows.

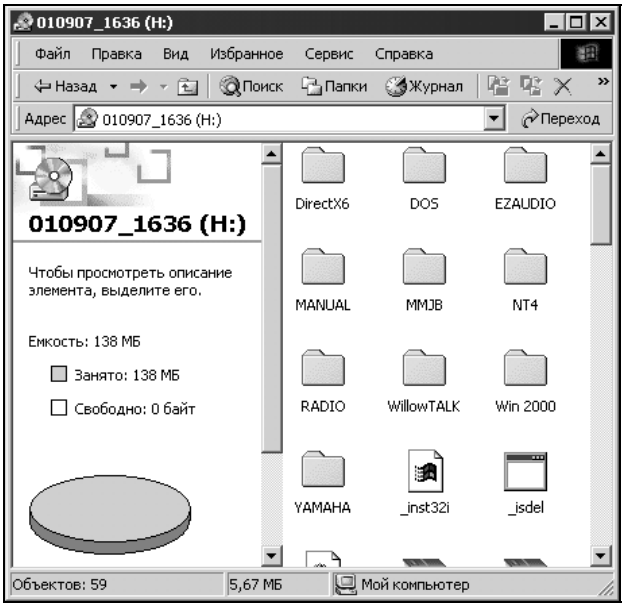


Рис. 5.10. Каталог диска, для которого на рис. 5.9 показан сектор

Из сравнения рис. 5.9 и 5.10 мы сразу заметим, что каталог начинается с названий папок. Однако после папки YAMAHA идут неизвестные имена: DirectX и далее названия языков ENGLISH, FRENCH и т. д. Оказывается, что это не что иное, как вложенные папки данного диска (отметим, что корневой каталог дискеты работает с папками не совсем так).

Обратим внимание читателей на еще одну интересную особенность, которая сразу бросается в глаза на рис. 5.9: в именах каталога между буквами регулярным образом расположены точки. Присмотревшись повнимательнее и сопоставив текст с соответствующими шестнадцатеричными кодами, можно заметить, что каждому символу соответствует два байта, причем первый из них везде нулевой. Например, выделенный символ **E** имеет код 00 45, а следующий за ним **Z** — 00 5A. Такое несколько необычное представление символов объясняется тем, что они хранятся в стандарте Unicode, согласно которому старший байт обозначает язык, а младший — код символа в нем. В нашем случае язык английский (латинские буквы), а ему для обеспечения совместимости со старым стандартом латинских букв присвоен нулевой код. Для русского языка установлен код 04.

Вывод. Итак, мы научились читать сектора и с CD-ROM. Как видно, принципы хранения секторов на магнитных и оптических дисках во многом похожи.

5.6.4. Форматирование дискеты

В ходе данного эксперимента мы познакомимся поближе с сущностью процесса форматирования.

Цель эксперимента. Выяснить, какие действия производятся в ходе форматирования диска.

Наверное, каждый пользователь Windows хотя бы раз видел диалоговое окно, предшествующее форматированию дискеты, изображенное для Windows 98 на рис. 5.11.

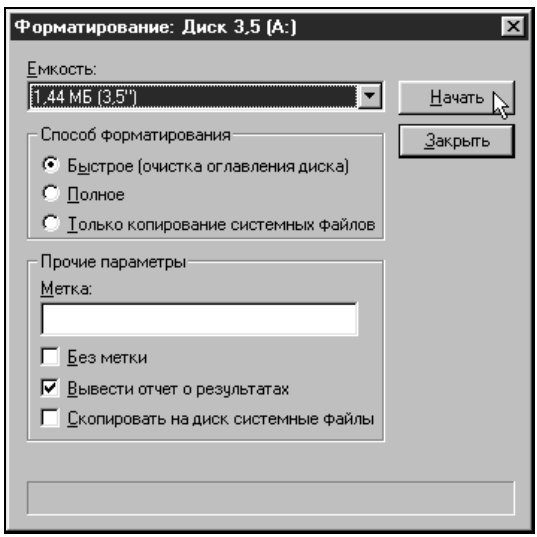


Рис. 5.11. Диалоговое окно форматирования дискеты

Из рис. 5.11 видно, что при форматировании дискеты решается несколько задач:

- нанесение на дискету разметки, т. е. служебной информации и "пустых" секторов, заполненных одинаковым кодом; указанные действия выполняются только при полном (физическом) форматировании;
- очистка каталога (оглавления) диска — логическое форматирование;
- занесение на дискету необходимой системной информации, после чего она становится загрузочной.

В наших экспериментах мы будем следить за несколькими областями дискеты. Это загрузочный сектор с номером 0, каталог диска, начинающийся с сектора 19, и собственно данные файлов, располагающиеся в секторе номер 33 и далее. Наиболее подготовленные читатели могут дополнительно контролировать состояние двух копий FAT (File Allocation Table) (сектора 1—9, 10—18). Подчеркнем, что все указанные номера секторов десятичные.

Начнем с полного форматирования дискеты. Выбрав соответствующий вариант, запустим его и терпеливо дождемся окончания процесса. Затем, используя одну из "заготовленных" в *разд. 5.6.1* программ, проверим содержимое нескольких секторов в характерных областях, перечисленных выше. Убедимся и запомним для себя, что пустой каталог заполнен нулями, а вместо информации везде занесен код F6.

Примечание

Если при форматировании указать метку диска, то она будет видна в первом секторе каталога.

Перекопируем на дискету несколько коротких файлов. Снова проверим каталог и найдем их там. Убедимся, что в области файлов тоже появилась некоторая информация. Теперь можно приступать ко второму эксперименту — быстрому форматированию. Проделайте его и проверьте, что информация в области файлов осталась нетронутой, а область каталога очистилась.

Наконец, если захотите опробовать все режимы, перенесите на диск систему и посмотрите, как это изменит системные области дискеты.

Совет

Удобно при сравнении содержимого секторов пользоваться программой sect_win, причем окна с полученным содержимым не закрывать.

Вывод. Таким образом, мы видели четкую разницу между полным и быстрым форматированием диска. Еще раз подчеркнем, что в случае сбоев дискеты полезно попытаться использовать именно полное форматирование.

5.6.5. Считывание S.M.A.R.T.-параметров жесткого диска

Прочитав *разд. 5.2.4*, многие читатели, возможно, захотят посмотреть S.M.A.R.T.-параметры своего жесткого диска. Это и будет целью нашего очередного эксперимента.

Цель эксперимента. Убедиться в существовании S.M.A.R.T.-информации и посмотреть, какие именно параметры она содержит. Особо подчеркнем, что в цели эксперимента не входит определение технического состояния жесткого диска, поскольку для этого требуется гораздо более систематическое изучение принципов данной технологии.

Для проведения эксперимента необходимо специально программное обеспечение. Можно воспользоваться, например, короткой программой SMARTUDM, загрузив ее из Интернета [И-27]. Программа проста в эксплуатации и включает хорошую документацию на русском языке, в том числе и по интересующим нас сейчас S.M.A.R.T.-параметрам (она написана М. Мавриципым).

Некоторым недостатком программы можно считать то, что она рассчитана на работу в среде MS-DOS и при запуске из-под Windows может не показать интересующие нас данные.

Результаты работы программы выдаются на экран и выводятся в виде протокола в текстовой файл. К сожалению, из-за наличия некоторых специфических псевдографических символов воспроизвести его в книге трудно. Поэтому будем предполагать, что читатели уже получили на своем компьютере копию протокола и посмотрели ее.

Количество наблюдаемых параметров жесткого диска достаточно велико. Приведем лишь несколько наиболее характерных из них, пользуясь имеющейся в документации расшифровке.

- Raw Read Error Rate (частота появления ошибок при чтении данных с диска). Показывает частоту появления ошибок при операциях чтения по вине аппаратной части.
- Soft Read Error Rate (частота появления "программных" ошибок при чтении данных с диска). То же, но для ошибок по вине программного обеспечения.
- Uncorrectable Sector Count (количество нескорректированных ошибок). Показывает общее количество ошибок, возникших при чтении/записи сектора, которые не удалось исправить. Рост значения параметра указывает на проблемы в работе накопителя.
- Reallocated Sectors Count (количество переназначенных секторов). О механизме "подмены" некачественных секторов уже упоминалось в *разд. 5.2.4*. Следует понимать, что при большом количестве переназначений хотя и сохраняется видимость не содержащего ошибок диска, но скорость доступа к информации замедляется.
- Current Pending Sector Count (текущее количество нестабильных секторов). Фактически речь идет о "кандидатах" в предыдущий параметр. При наличии новых ошибок в работе с таким сектором, он будет переназначен. Постоянное ненулевое значение этого атрибута говорит о низком качестве поверхности диска или ее части.
- Power-On Hours (количество отработанных во включенном состоянии часов). Этот и следующий параметр понятны без пояснений.
- Device Power Cycle Count (количество полных циклов запуска/останова жесткого диска).
- Drive Temperature (температура диска). Измеряется встроенным в накопитель датчиком.
- G-Sense Error Rate (частота появления ошибок в результате ударных нагрузок). Для измерения этого параметра также предусмотрен специальный датчик. Так что все ударные нагрузки на дисковод фиксируются!

Существуют также программы для индикации S.M.A.R.T.-параметров и для среды Windows. Вот как, например, выглядит одна из них [И-20] (рис. 5.12).

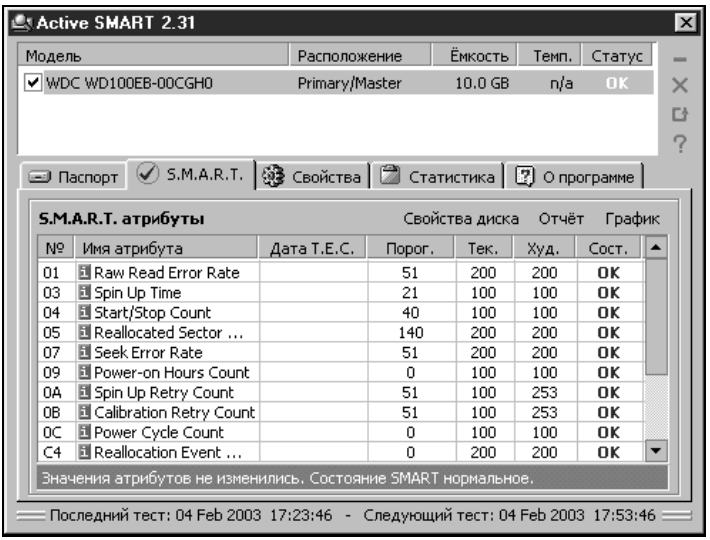


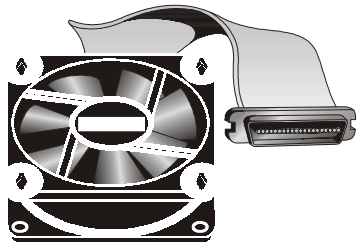
Рис. 5.12. Программа-монитор для контроля S.M.A.R.T.-параметров жесткого диска

Важно

Программы контроля S.M.A.R.T.-параметров обычно способны копить и анализировать статистику изменения параметров с целью прогнозирования технических неполадок жесткого диска. Но для того, чтобы это предсказание имело смысл, период накопления статистики должен быть как минимум несколько месяцев. Поэтому если в ходе наших экспериментов программа-монитор сначала пообещала "крах" вашего накопителя через пару месяцев, а на следующий день бесследно ликвидировала предупреждение, не обращайте на все это внимание.

Вывод. Таким образом, эксперимент показывает, что S.M.A.R.T.-информация действительно достаточно подробно характеризует техническое состояние жесткого диска.

ЛЕКЦИЯ 6



Устройства ввода/вывода

Надеюсь, читатели еще не устали от обсуждения того многообразия устройств, которые можно подсоединить к современному компьютеру. Наберитесь, пожалуйста, терпения, в данной лекции мы рассмотрим последнюю их группу, к тому же самую эффектную — устройства ввода/вывода. Именно с их помощью мы вводим информацию в ЭВМ и получаем от нее результаты обработки данной информации. И если усовершенствование модуля памяти или винчестера может быть не всегда сразу заметно, то любые изменения в устройствах ввода/вывода немедленно бросаются в глаза даже малоопытным пользователям.

Несколько слов об особенностях *лекции 6*. При изложении ее материала предполагается, что читатель обладает хотя бы минимальными представлениями об общении с компьютером: видел когда-нибудь монитор и клавиатуру, а также знает, для чего на мыши есть кнопки и с какой целью к компьютеру подключают печатающее устройство. Автору кажется, что в настоящее время компьютеры уже распространены настолько широко, что вряд ли целесообразно уделять внимание подобным очевидным вопросам. И хотя многие популярные книги и даже некоторые учебники по-прежнему придерживаются противоположной точки зрения, посвящая основную часть описания внешнему виду тех или иных устройств и их тривиальным функциям, такой подход явно противоречит принципам нашего курса, сформулированных в *разд. 1.2*. По той же причине лекция не сопровождается фотогалереей внешнего вида различных устройств ввода/вывода, хотя, возможно, само по себе это и было бы неплохо.

Итак, поговорим об этих важных устройствах подробнее.

6.1. Назначение и виды устройств ввода/вывода

Назначение данной категории устройств достаточно очевидно — осуществлять обмен информацией между человеком и ЭВМ. Благодаря им пользова-

тель может ввести в компьютер данные для обработки и увидеть ее результаты. Кроме того, возможно, не все задумывались над тем фактом, что сама программа обработки информации также первый раз (имеется в виду, ее разработка) попадает в память машины через одно из устройств ввода; лишь после этого ее (или лучше ее эквивалент в машинном коде) можно сохранить во внешней памяти и в дальнейшем считывать, например, с диска, не повторяя набора. Таким образом, вся информация первоначально попадает в компьютер через устройства ввода, а все доступные человеку результаты появляются из устройств вывода.

Современные устройства ввода/вывода довольно разнообразны как по принципам работы, так и по видам информации, с которыми они работают. Прошли времена, когда набор текста был единственным вариантом ввода, а печать на бумагу чисел с короткими комментариями — единственным способом представления результатов. Современные компьютеры уже имеют устройства для ввода рукописной и голосовой информации и могут выводить результаты на слайды или в виде заготовок для книгопечатания. Разнообразие устройств настолько велико, что даже по их отдельным группам издаются целые книги (см. [12]).

6.1.1. Устройства ввода

Самыми первыми устройствами ввода в ЭВМ были различные перемычки и переключатели. С помощью этого естественного для машины (и, смею заметить, довольно противоестественного для человека) способа, когда замыкание цепи обозначало единицу, а его отсутствие — ноль, информация в двоичном виде и вводилась в ЭВМ. Такое неудобное общение не могло продолжаться долго; поэтому, как только машины перестали быть объектом лабораторных исследований ученых и начали решать практические задачи, инженеры сразу же предложили для ввода более комфортабельное решение — электрифицированную пишущую машинку. При нажатии каждой ее клавиши на выходах такого устройства появлялся вполне определенный двоичный код, который и поступал в ЭВМ. Кроме того, на бумаге автоматически печатался протокол набора, что позволяло впоследствии его проверить и исправить допущенные "опечатки". В некоторых ЭВМ вывод результатов происходил на ту же самую пишущую машинку, так что весь диалог оставался зафиксированным на бумаге.

По мере усложнения диалога оператора с ЭВМ расход бумаги становился все более значительным. Особенно это было неприятно в тех случаях, когда результат по тем или иным причинам получался неудовлетворительным. Естественно, что возникло стремление полностью разделить процессы ввода обрабатываемой информации в машину, ее отображение и печать конечных результатов на бумагу. С изобретением дисплеев это стало возможным, и клавиатура превратилась в "чистое" устройство ввода, что существенно ее упростило. Появление такого рода клавиатур следует, видимо, связывать с третьим поколением ЭВМ.

Многие начинающие пользователи, не задумывающиеся над тем, что происходит при вводе, считают, что клавиатура и отображающий вводимый текст дисплей связаны напрямую, т. е. как только мы нажимаем клавишу, ее код тут же поступает в монитор и отображается. Это далеко не так. На самом деле сначала код нажатой клавиши поступает в компьютер, который программно его анализирует и затем по результатам этого анализа выводит на экран монитора тот или иной символ или группу символов. Только так можно объяснить, почему от одной и той же клавиши на экране возникает русская или латинская, а также заглавная или строчная буква. Еще более убедительно выглядит нажатие служебных, например, функциональных клавиш, когда на экране появляются целые слова или вообще производится вызов страницы с текстом помощи. Таким образом, запомним, что процесс ввода данных с клавиатуры является вполне самостоятельной операцией.

Внутреннее устройство клавиатур может быть довольно разнообразным. Например, они могут использовать замыкание металлизированного слоя, нанесенного на гибкую основу, работать на базе герконов (*гер*метичных *кон*тактов, которые замыкаются при поднесении магнита) или фиксировать возникающее при нажатии клавиш изменение емкости. Внешнее оформление клавиатур также может существенно варьироваться.

Совершенствование устройств ввода тесно связано с развитием диалога человека и ЭВМ. Так переход от текстового способа общения с компьютером (набор команды) к графическому (выбор пиктограммы или пункта меню) вызвал появление принципиально нового устройства ввода — манипулятора "мышь". Интересные данные о происхождении мыши можно найти в [79, И-19].

Казалось бы, что сложного в этом миниатюрном устройстве, габариты которого определяются в основном формой и размерами руки человека. Наверное, каждый, кто хоть немного интересуется вычислительной техникой, разбирал старую мышь. Шарик, вращение которого передается на два взаимно перпендикулярных колесика с прорезями, при вращении периодически перекрывающими световой поток. Но между прочим, задумывались ли вы когда-нибудь, как мышь определяет направление вращения колесиков и, следовательно, направление движения самой мыши? Попробуйте порассуждать, и вы убедитесь, что в силу симметрии механической системы единственный датчик принципиально не в состоянии дать ответ на этот вопрос! Оказывается, что для решения данной проблемы в устройстве обязательно используются два (!) датчика, хотя они часто и располагаются в одном корпусе. В качестве доказательства взгляните на замечательные фотографии внутреннего устройства мыши, которые приводятся на сайте [И-21][1].

---

[1] Хотя данный сайт под названием "Как работают вещи" ведется на английском языке, автор убедительно рекомендует его посетить.

В последнее время индустрия производства мышей не перестает удивлять пользователей различными усовершенствованиями. Появились оптические мыши, устроенные настолько интересно, что просто невозможно пройти мимо данного вопроса. Оптическая мышь не содержит никаких движущихся частей — их заменяет миниатюрная видеокамера (!), которая более тысячи раз в секунду делает снимки поверхности "под брюхом" мыши [И-21, И-17]. Для анализа полученных изображений используется специальный микропроцессор (так называемый DSP — Digital Signal Processor), способный за секунду выполнить 18,5 миллионов операций. Подобная вычислительная мощь позволяет ему сопоставлять два последних снимка и по характерным деталям изображения определять направление и величину сдвига мыши. Для освещения поверхности используется небольшой красный светодиод (эффектный снимок, демонстрирующий эту подсветку, опубликован в [44]). За редкими исключениями, каковыми являются зеркало, стекло и бархат, мышь прекрасно работает на плоской поверхности из любого материала. По утверждениям производителей, человеческая рука принципиально неспособна передвинуть мышь так быстро, чтобы последняя "не успела" корректно отследить движение.

Небольшое лирическое отступление

Каждый лектор, даже самый серьезный, нет-нет да и отойдет от изложения материала, чтобы рассказать забавную историю из жизни известного ученого или описать свое личное отношение к той или иной проблеме. Поэтому смею надеяться, читатели правильно воспримут данный абзац. Мне кажется, что если вас не восхищает описанное только что устройство мыши, то вряд ли вам вообще будет по-настоящему интересна информатика, как бы хорошо вы не рисовали схемы в графическом редакторе и не находили информацию в Интернете. Я далек от мысли, что быть равнодушным к устройству ЭВМ ненормально или плохо; тем не менее мне кажется странным, что многие люди с подобными взглядами не могут понять, что бывает иначе. К сожалению, у меня нет статистики, какой процент людей, использующих компьютер, интересуется его устройством. Но такие люди определенно есть, и их немало. Вот, например, абсолютно независимая цитата с точно такой же, как у меня, оценкой: "Ее принцип действия меня просто потряс — оптический датчик в корпусе мыши делает мгновенный снимок поверхности под ней, а затем специальный ... процессор ... сравнивает два последовательных снимка, чтобы вычислить смещение. Это ж надо было такое придумать". [И-2] И еще. Посмотрите, что обычно публикуют в русском сегменте Интернета по поводу тех же мышей: перечисление фирм производителей, стоимости, сравнение мягкости нажатия кнопок и точности прицела при расстреле всего, что движется, в пресловутом DOOMe... Как тут не вспомнить цитату из Бредбери, приведенную в *разд. 1.1*? Ладно, достаточно, вернемся к нашим мышам.

Для удобства работы пришлось "пожертвовать" даже характерным "хвостом" мыши, т. е. кабелем, соединяющим ее с компьютером. Беспроводные мыши передают данные о своем движении с помощью инфракрасного сигнала, подобно тому, как это делается в пультах дистанционного управления теле-

визорами. А еще существует мышь, которая при наведении курсора на пиктограмму, пункты меню, ссылку или что-нибудь подобное, отзывается дрожанием встроенного моторчика, причем вибрация будет разного типа в зависимости от объекта. Экзотические виды мышей и клавиатур можно посмотреть на Web-страничке [И-10].

Ближайшим "родственником" мыши является трэкбол. Это своеобразная "перевернутая" мышь, у которой шарик находится сверху. Еще один манипулятор — джойстик — предназначается главным образом для игр.

Другая причина, которая приводит к расширению ассортимента устройств ввода — это обработка на компьютере новых типов информации. Потребности во вводе графической информации породили дигитайзер (устройство для ручного ввода в ЭВМ рисунков или чертежей [35]) и сканер [12], а работа со звуком и видео потребовала подключения к компьютеру микрофона и видеокамеры. И если для записи звука годится обычный микрофон, например, имеющийся в комплекте магнитофона, то видеокамеры для компьютера были разработаны особые — цифровые [12]. К сожалению, профессиональные цифровые камеры пока довольно дóроги, зато для общения в Интернете, где все равно нельзя передать качественное изображение из-за его большого объема, выпускаются специальные недорогие, хотя и менее качественные, Web-камеры.

Примечание

В начале данного абзаца не случайно сказано, что сканер вводит только графическую информацию. Часто используемое на практике преобразование полученного изображения в текст осуществляется не сканером, а самим компьютером по специальной программе.

Таким образом, мы видим, что развитие вычислительной техники приводит к существенному увеличению разнообразия устройств ввода. Этому, прежде всего, способствует рост возможностей компьютера и расширение тех видов информации, которые он обрабатывает.

6.1.2. Устройства вывода

Поговорим теперь об устройствах вывода. К самым первым устройствам такого рода следует, видимо, отнести "линейки" индикаторных лампочек, которые отображали состояние регистров ЭВМ. Опытному человеку (а другим тогда за пультом просто нечего было делать!) эти комбинации нулей и единичек могли рассказать о многом.

Устройства для вывода результатов расчетов на бумагу были уже в ЭВМ первого поколения. Их называли ЦПУ, т. е. цифровые печатающие устройства, поскольку кроме столбика цифр они ничего не могли напечатать. Довольно быстро им на смену пришли более совершенные АЦПУ (алфавитно-цифровые печатающие устройства), способные кроме чисел выводить на

бумагу всевозможные текстовые комментарии. Подобные устройства имели огромные габариты и вес, при печати производили сильный шум, но зато могли на широком листе бумаги выводить по несколько строк за одну секунду.

Все упомянутые устройства, включая электрифицированные пишущие машинки, о которых шла речь в предыдущем разделе, могли печатать только символы, причем не произвольные, а лишь те, шаблоны которых были заложены в их конструкцию (отлиты на металлическом барабане или рычагах литер, ударяющих через красящую ленту по бумаге). Важным шагом вперед в расширении возможности печатающих устройств явился переход к так называемому матричному принципу формирования символов. Суть его состоит в том, что печатающая головка содержит вертикальный ряд специальных иголок (от 8 до 24 в более поздних моделях), каждая из которых управляется автономно. Благодаря движению головки вдоль листа бумаги принтер получает возможность сформировать изображение символа из отдельных точек. Гибкая система управления обеспечивает возможность печатать символы с любыми начертаниями, в том числе те, которые определил сам пользователь. Нетрудно сообразить, что фактически это уже не устройство для вывода текста, а простейшее графическое устройство, способное выводить на бумагу схемы, диаграммы и даже не очень сложные рисунки.

В дальнейшем технология изготовления печатающих устройств была существенно усовершенствована, и появились струйные и лазерные принтеры. В основе работы первых лежит управление тончайшими струйками чернил (например, электростатическим способом), которые попадают на бумагу и создают требуемое изображение. Вторые имеют специальный светочувствительный барабан, на котором луч лазера создает скрытое электростатическое изображение; от полученного распределения заряда зависит, сколько прилипнет на вал порошка (тонера) и, следовательно, как много его будет перенесено на бумагу; окончательная фиксация полученного на листе изображения осуществляется термическим методом.

Устройство струйных и лазерных печатающих устройств является одной из любимых тем популярной литературы о компьютерном оборудовании. За более подробными сведениями по этому вопросу можно, например, обратиться к уже упоминавшейся книге [12] или обзору [И-5]. В частности, прочитав указанную книгу, вы сможете дополнительно узнать, что существует еще твердокрасочная печать с помощью восковых красителей, расплавляемых при нанесении на носитель, а также сублимационные (термодиффузионные) принтеры, работающие очень медленно, но зато обеспечивающие необычайно высокое качество цветопередачи. Нет сомнения, что список этот будет постоянно пополняться; вполне возможно даже, что за время, прошедшее от момента написания этих строк до того момента, когда вы их читаете в книге, уже появилась какая-то новая технология печати.

Параллельно с процессом улучшения качества вывода информации на бумагу развивались и другие компьютерные устройства вывода. Принципиально

важным шагом, как уже говорилось ранее, было перенесение рабочего диалога человека и ЭВМ с бумаги на экран дисплея. Как вы, конечно, знаете, специфика работы с компьютером такова, что очень часто фиксировать нужно лишь окончательный результат, а все промежуточные действия (набор и редактирование текста, процесс создания рисунка и т. п.) нет особой необходимости документировать. Наличие "безбумажного" дисплея также позволило существенно повысить комфортность работы пользователя за счет всевозможных виртуальных органов управления, меню, окон, подсказок и т. д.

Примечание

Мы настолько привыкли к работе с монитором, что даже не задумываемся о том, какие сложные действия производит машина, формируя изображение на экране дисплея. Попытайтесь для интереса мысленно представить себе процесс формирования экрана среды Windows на бумаге, причем именно поэтапное формирование, а не снятие копии готового итогового изображения! Помоему, у вас просто ничего не получится, хотя бы потому, что уже выведенное на бумагу изображение нельзя удалять!

Первые дисплеи появились на завершающем этапе второго поколения. В частности, дисплей играл большую роль в работе отечественной ЭВМ МИР-2. Интересно, что это красивое название на самом деле расшифровывалось весьма прозаически — Машина для Инженерных Расчетов; фотографию машины можно посмотреть в виртуальном музее по адресу [И-7]. В третьем поколении дисплей оператора был уже одним из ведущих устройств управления. Отметим, что благодаря своему принципу работы, согласно которому изображение строится из отдельных точек, чисто текстовые дисплеи достаточно быстро были вытеснены графическими.

Довольно долго дисплеи конструировались на базе электронно-лучевой трубки, во многом напоминающей кинескоп обычного телевизора (не случайно раньше некоторые простейшие бытовые компьютеры просто подсоединялись к телевизору). В последнее время появились принципиально новые способы построения мониторов [41, 42].

Как и для рассмотренных ранее устройств ввода, на расширение набора устройств вывода оказывает большое влияние работа с новыми видами информации. Например, для вывода звуков и музыки созданы специальные звуковые платы. По сравнению с большинством других устройств ввода/вывода они имеют существенную особенность: результатом их работы является непрерывный, а не дискретный сигнал. Специальный преобразователь (его принято называть ЦАП, т. е. *цифроаналоговый преобразователь*) обеспечивает плавное изменение выходного напряжения, которое подается на внешние громкоговорители. Для получения высококачественного эффекта лучшие платы используют сглаживание формы сигнала по 15—20 последовательным дискретным значениям, хранящимся в звуковом файле [72].

Как видят читатели из нашего короткого обзора (а в нем многие интересные устройства, к сожалению, лишь упомянуты), устройства ввода/вывода и их конкретные принципы реализации настолько многообразны, что заслуживают отдельного разговора в отдельной книге, на что автор совершенно не претендует. Поэтому в следующем разделе мы перейдем к обсуждению наиболее общих принципов обмена информацией с устройствами ввода/вывода, справедливых для любого из них.

6.2. Организация ввода/вывода

Позволю себе напомнить читателям, что согласно фундаментальному магистральному принципу устройства компьютера *(см. разд. 2.2.3)*, все устройства ввода/вывода, равно как и устройства внешней памяти, взаимодействуют с общей шиной одинаковым образом. Следовательно, все обсуждаемое в данном разделе можно отнести к любому из этих периферийных устройств.

6.2.1. Порты

Способ приема данных или передачи их внешним устройствам существенно зависит от конструкции ЭВМ (подчеркнем: не от типа микропроцессора, а от организации всей ЭВМ!). А конструкция эта может быть реализована одним из двух указанных ниже способов:

- устройства ввода/вывода включаются в общее адресное пространство внутренней памяти;
- устройства ввода/вывода имеют собственное адресное пространство.

В первом случае при обращении к определенным адресам памяти вместо обмена с ОЗУ происходит аппаратное подключение того или иного внешнего устройства. При этом для "общения" с внешними устройствами и с памятью используются одни и те же команды процессора, хотя, конечно, обмен с внешним устройством протекает по более сложному алгоритму, чем с памятью.

Во втором случае внешние устройства образуют отдельное адресное пространство, обычно значительно меньшее, чем у ОЗУ. Каждая "ячейка" этого дополнительного адресного пространства называется *портом*. Каждому внешнему устройству обычно соответствует несколько портов с последовательными адресами. Обмен процессора с организованными подобным образом устройствами осуществляется с помощью специальных команд ввода/вывода.

На самом деле, с точки зрения схемной организации оба описанных способа имеют очень много общего. При любом обмене, будь то обращение к ОЗУ, ПЗУ или внешнему устройству, процессор выставляет адрес информации на единую адресную шину, а данные передает или принимает по общей

шине данных. Выбор же требуемого адресата — ячейка памяти или порт — осуществляется подачей специального сигнала по шине управления.

В одном и том же компьютере могут одновременно встречаться оба способа адресации устройств ввода/вывода. Например, не так давно в системе нашего образования были распространены замечательные учебные компьютерные классы Yamaha японского производства. Так вот в них накопитель на магнитных дисках был включен в общее адресное пространство памяти, а печатающее устройство оформлено в виде нескольких портов.

Примечание

Любопытно, что некоторые порты этого компьютера служили для выбора и подключения к небольшому 64-килобайтовому адресному пространству 128 Кбайт ОЗУ и многочисленных ПЗУ. При этом все ресурсы памяти были разбиты на отдельные страницы по 16 Кбайт и в зависимости от содержимого управляющих портов в каждый момент времени подключались требуемые четыре из них.

6.2.2. Обмен по программе

Рассказ о работе с периферийными устройствами был бы неполным без описания того диалога, который ведет с ними процессор. Рассмотрим его на примере простейших устройств ввода/вывода, например, печатающего устройства.

Обмен с подключенным к ЭВМ печатающим устройством производится через три порта — порт данных, порт состояния и порт управления. В первый микропроцессор помещает данные для вывода на бумагу, во втором хранится информация о состоянии устройства в данный момент времени, а через третий осуществляется "руководство" процессом вывода данных. Каждый бит порта состояния хранит ответ на вполне определенный вопрос: заправлена ли бумага, готов ли принтер принять данные от компьютера и т. д. Для процессора все это входные сигналы. Но есть и выходные, которые через порт передаются к печатающему устройству. Наиболее важным из них является бит, свидетельствующий о готовности информации в порту данных к передаче: через этот бит процессор оповещает принтер, что данные сформированы и их можно считывать. Такой управляющий сигнал в специальной литературе называется *стробом*.

В наиболее простом случае обмен информацией между процессором и принтером может протекать следующим образом. Пусть микропроцессор (МП) должен вывести на печать какой-нибудь символ. Он считывает порт состояния принтера и анализирует содержимое его бита готовности. Если результат положительный, т. е. печатающее устройство готово принять информацию, обмен продолжается, в противном случае МП снова считывает порт состояния и повторяет анализ.

Когда процессор получил от принтера сигнал о готовности к обмену, он заносит требуемый символ в порт данных и установкой стробирующего сигна-

ла информирует об этом принтеру. Затем процессор снова периодически считывает порт состояния, но следит уже за другим битом: через этот бит принтер сообщит процессору о том, что данные приняты, т. е. скопированы в собственное ОЗУ принтера. После этого МП убирает стробирующий потенциал и продолжает работу по программе, а принтер в это время выводит принятый символ на бумагу.

Конечно, реальный диалог современного компьютера с печатающим устройством несколько сложнее. Вы, наверное, заметили, что описанный выше алгоритм "зациклится", если принтер не подключен или не готов к работе: процессор будет ждать сигнала о готовности принтера, а того не будет. Выход из создавшегося положения — ожидать сигнала готовности в течение какого-то времени, а затем выдать пользователю сообщение о неготовности принтера к работе. При этом желательно проверить содержимое других битов порта состояния (например, отвечающих за наличие бумаги и т. п.) и уточнить, если потребуется, диагностическое сообщение.

В *разд. 6.7.2* вопросы взаимодействия принтера с машиной обсуждаются подробнее в связи с проведением одного из экспериментов.

Думается, прочитав даже достаточно поверхностное описание механизма обмена процессора с таким относительно простым устройством, как принтер, мы должны еще раз проникнуться уважением к тем программистам, которые избавили нас от необходимости учитывать все эти тонкости и позволили нам просто выбрать в соответствующем меню пункт "печать".

Аналогичный диалог происходит у процессора и с другими внешними устройствами. Если вам захочется для сравнения рассмотреть еще какое-либо устройство, вы можете обратиться к [И-12], где хорошо сформулирован алгоритм взаимодействия центрального процессора и жесткого диска.

6.2.3. Обмен по прерываниям

Обмен информацией между процессором и устройством ввода следует рассмотреть отдельно, т. к. он может иметь некоторые существенные особенности. Описанная выше идеология передачи данных, конечно, может быть использована и для устройств ввода, например для клавиатуры. В этом случае программа, в тот момент, когда ей потребуются входные данные, опросит состояние клавиатуры и примет с нее данные. Однако для современных программных систем такой метод неудобен из-за существенного замедления времени реакции ЭВМ. В самом деле, если некоторая программа занята трудоемкими вычислениями, а вы захотите ее прервать, то придется подождать, пока процессор освободится и займется опросом клавиатуры (собственно говоря, тогда и прерывать-то будет нечего!) Не правда ли, все это сильно напоминает очередь в приемной бюрократа-начальника, когда люди вынуждены часами ждать, пока их вызовут для пятиминутного решения вопроса?!

Для предотвращения подобных неприятностей, во всех современных ЭВМ наряду с опросом устройств ввода по программе существует еще один механизм — *прерывания*. Прерывания настолько широко используются в компьютерах, что имеет смысл рассмотреть их подробнее.

Для понимания сути работы прерываний снова обратимся к аналогии с руководителем. Пусть в его кабинете идет совещание по подведению итогов деятельности предприятия, и в этот момент по телефону поступает очень важная информация, требующая немедленного принятия решения. Как в таком случае обычно развиваются события? Секретарша, не дожидаясь конца совещания, сообщает шефу о звонке. Тот, прервав свое выступление, снимает трубку и выясняет суть дела. Затем он либо тут же принимает решение и сообщает его, либо предлагает пока сделать самое необходимое, а после совещания обещает перезвонить и дать дальнейшие указания. Потом (обратим внимание на эту деталь!) он произносит что-нибудь в духе "Так на чем мы остановились?" и продолжает совещание, как ни в чем не бывало. Впрочем, если ситуация экстренная, то совещание может быть прекращено или в его ход могут быть внесены определенные коррективы... Да простит читатель столь длинное отступление, но оно очень полезно для понимания сути механизма прерываний.

Вернемся теперь к компьютеру. Каждое его устройство (клавиатура, мышь, дисковод и др.) способно затребовать внимания процессора, выставляя электрический сигнал требования прерывания. Процессор проверяет наличие этого сигнала после выполнения каждой операции; вы, вероятно, помните, что в *разд. 3.2.1* автор обещал добавить к основному алгоритму работы процессора еще один этап — так вот это и есть тот самый обещанный этап! "Увидев" требование прерывания, процессор немедленно начинает его обрабатывать. Прежде всего, он запоминает свое текущее состояние — содержимое счетчика команд и регистра состояния — в стек *(см. разд. 4.3.3)* с тем, чтобы в дальнейшем иметь возможность продолжить выполнение прерванной программы. После этого происходит переход на программу, обрабатывающую данное прерывание. Заметим, что все перечисленные действия реализуются на аппаратном уровне, т. е. фактически являются "врожденными рефлексами" микропроцессора.

Программа обработки прерывания, получив управление, проводит оперативный анализ причины прерывания. Если причина серьезная и требует немедленных действий, то они выполняются. Если же с обработкой можно подождать (например, просто нажата обычная символьная клавиша), в специальную область памяти — *буфер* — заносится информация о происшедшем прерывании. Во всех случаях прерывания завершаются выполнением специальной команды возврата, которая восстанавливает содержимое программного счетчика и регистра состояния, давая тем самым микропроцессору возможность "как ни в чем не бывало" продолжить работу прерванной программы.

Реальная картина, конечно, несколько сложнее: прерывания от разных устройств имеют разные уровни приоритета, существует возможность маскировки отдельных прерываний (т. е. переключения процессора в режим, когда он такой вид прерываний не замечает), программа обработки прерывания сохраняет и восстанавливает значения используемых ей регистров МП и т. п. К счастью, сути дела все это существенно не меняет.

Примечание

Обсуждаемый механизм аппаратных прерываний от внешних устройств не следует путать с программными прерываниями. Например, для IBM-совместимых компьютеров в литературе часто описываются многочисленные команды прерывания `INT` с самыми разнообразными номерами. Следует четко представлять, что `INT` — это одна из инструкций процессора; чтобы она заработала, ее код должен содержаться в программе. В противоположность этому, "настоящие" прерывания возникают аппаратно и не требуют наличия каких-то специальных команд в тексте прерываемой программы. Более того, аппаратное прерывание может произойти между двумя любыми командами программы.

В последнее время необходимость понимания механизма работы прерываний сильно возросла в связи с возникновением идеологии программирования по событиям. Она лежит в основе последних систем типа Visual Basic (напомним читателям, что на базе этого языка построен Microsoft Office) или Delphi. Приведем примеры нескольких событий, на которые программа может реагировать: сдвинута мышь, нажата (или отпущена) ее кнопка, нажата клавиша <Enter>, выбран тот или иной пункт меню, открыто новое окно на экране и многие-многие другие. Полный перечень событий занимает в описании несколько страниц. Интересно, что программа на таком языке является единым целым только формально: на каждое событие фактически пишется своя собственная подпрограмма, хотя, разумеется, все они и могут взаимодействовать между собой.

Пример простейшей обработки событий продемонстрирован в *разд. 6.7.4*.

6.3. Необходимость программной настройки устройств

Многообразие внешних устройств и наличие у них многочисленных фирм-производителей, не может не создавать некоторые сложности при подключении к компьютеру новых устройств. Хотя электрические параметры компьютерных плат и являются достаточно стандартными, тем не менее отдельные мелкие детали работы все же различаются. Окончательная подстройка определенных устройств конкретного производителя в данном компьютере обычно производится программным путем. Специализированные программы для управления работой внешних устройств принято называть *драйверами*.

В развитых странах при покупке любого внешнего устройства вам обязательно дадут дискеты или CD-ROM с программной поддержкой для этого устройства. Уважающая себя (и своих покупателей) фирма непременно постарается максимально облегчить пользователю установку своей продукции путем создания всевозможных программ автоматической инсталляции, которые задают пользователю минимально возможное количество достаточно простых вопросов, а всю остальную часть работы по настройке оборудования берут на себя. К сожалению, по целому ряду причин в нашей стране картина далеко не такая безоблачная, поэтому установка нового устройства порой превращается в мучительный процесс проб и ошибок с неоднократной перезагрузкой компьютера.

Справедливости ради следует подчеркнуть два очень важных обстоятельства, которые заметно облегчают описываемый процесс установки новых устройств в современный компьютер. Во-первых, это существование Интернета, благодаря которому мы можем в любой момент обратиться непосредственно на Web-страницу производителя оборудования за самой последней версией драйверов. Во-вторых, практически все выпускаемые сейчас периферийные устройства соответствуют стандарту Plug and Play (приблизительно переводится на русский язык как "подключил и работай"). Главной особенностью таких плат является возможность их автоматической электронной настройки в ходе диалога с компьютером в процессе начальной загрузки. Читатели, заинтересовавшиеся работой стандарта PnP, могут почитать энциклопедию [15], где механизмы "самонастройки" оборудования компьютера описаны подробно и понятно.

Настроенный однажды компьютер при неизменных условиях эксплуатации теоретически не должен требовать повторения процедуры настройки еще раз.

6.4. Об объединении компьютеров в сеть

Работа на компьютере сейчас уже немыслима вне компьютерных сетей. Доступ к ним позволяет существенно повысить эффективность обработки данных, добавляя одиночной машине целый ряд дополнительных возможностей:

- использование общих устройств, подключенных к группе компьютеров (например, общий высококачественный принтер);
- возможность обрабатывать свою информацию на более мощных компьютерах, доступных по сети;
- доступ к данным, хранящимся на других компьютерах независимо от их географического местоположения;
- существенное повышение эффективности при коллективной работе над проектом;
- оперативный обмен информацией и почтой с другими пользователями.

Для работы с сетью необходимы специальные устройства. В случае, когда компьютеры объединены в локальную сеть, таким устройством является сетевая плата, обеспечивающая контакт с аналогичными платами в других компьютерах. Другой вариант позволяет подсоединить один или несколько компьютеров к сети с помощью стандартного телефонного канала. Для этого требуется более сложное устройство, называемое *модемом*; указанное название происходит от объединения слов *мо*дулятор и *демо*дулятор. Особенностью такого способа доступа является временное преобразование (модуляция) цифровых данных при передаче по аналоговому (нецифровому) каналу с последующим обратным преобразованием (демодуляцией). Кстати, довольно своеобразно и интересно описана работа модема в учебнике [52].

По мнению автора, устройство компьютерных сетей лучше рассматривать в отдельном курсе, поскольку при их проектировании как раз принимаются все возможные меры, маскирующие особенности работы отдельных машин, образующих сеть. Иными словами, преследуются цели, во многом противоположные таковым для нашего курса. Пытаясь предупредить возможные упреки, подчеркну, что данное обстоятельство ни в коем случае не свидетельствует об отрицании автором необходимости изучения компьютерных сетей и не умаляет важности освоения сетевых технологий. Напротив, это обязательно надо делать, причем именно отдельно, а значит, более подробно и обстоятельно.

6.5. Выводы

Вот мы и рассмотрели последнюю группу устройств компьютера. Казалось бы, этого должно быть достаточно для понимания основных принципов его работы. Но это не совсем так. Аппаратура (в иностранной литературе ее обозначают общепринятым термином "hardware") составляет лишь одну из составляющих современной ЭВМ. Другой, не менее важной частью, является программное обеспечение ("software"); ее мы подробно обсудим в следующей лекции.

А пока по традиции сформулируем выводы, которые следуют из изложенного материала.

- Рассмотренная категория устройств предназначена для обмена информацией между ЭВМ и человеком. Вся информация первоначально попадает в компьютер через устройства ввода, а все доступные человеку результаты появляются из устройств вывода.
- Современные устройства ввода/вывода довольно разнообразны как по принципам работы, так и по видам информации, с которыми они работают. Их ассортимент постоянно растет в связи с совершенствованием и расширением способов общения человека с компьютером и автоматизацией обработки все новых и новых видов информации.

- Устройства ввода/вывода и устройства внешней памяти имеют одинаковые принципы обмена данными через общую шину.
- Устройства ввода/вывода могут иметь собственное адресное пространство, ячейки которого называются портами, или "проектироваться" в адресное пространство внутренней памяти. Выбор того или иного способа зависит от конструкции конкретной ЭВМ, причем возможно сочетание обоих методов.
- Обмен между периферийными устройствами и центральным процессором может протекать как по инициативе последнего (т. е. по программе), так и по требованию внешних устройств (по прерываниям). В последнем случае современные микропроцессоры используют специальный встроенный механизм обработки прерываний.
- Не следует путать аппаратные прерывания со специальными командами, которые называются тем же самым словом; различие между ними является принципиальным.
- Помимо аппаратного подключения к компьютеру, каждое внешнее устройство нуждается в программной поддержке. Программы, обеспечивающие взаимодействие процессора с периферийными устройствами, называются *драйверами*.
- В современных компьютерах развивается и совершенствуется система автоматической настройки внешних устройств.
- Для обмена информацией между компьютерами также требуется специальное оборудование: сетевые платы или модемы.
- Современный компьютер является своеобразным "сплавом" программной и аппаратной части. Следует четко понимать, что без программной поддержки использовать его практически невозможно. Роли программного обеспечения в работе компьютера будет посвящена следующая лекция.

6.6. Вопросы для осмысления

1. Какие устройства ввода/вывода присоединены к вашему компьютеру? Какую информацию они могут вводить или выводить? Вспомните, приходилось ли вам видеть еще какие-нибудь устройства?
2. Знаете ли вы, на каком принципе работают ваши устройства ввода/вывода (например, какой у вас принтер — матричный, лазерный или струйный)?
3. Проследите эволюцию устройств ввода и вывода. Сможете ли вы предсказать, какие новые устройства могут появиться в ближайшее время? Что вы читали о последних разработках в этой области?
4. Как вы представляете себе процессы, происходящие в компьютере при наборе на клавиатуре слова?

5. Докажите, что, пользуясь единственным датчиком, легко установить факт вращения колеса с прорезями и даже его скорость, но невозможно определить направление вращения.
6. Каким образом можно реализовать ввод текста, используя обычную мышь? Попробуйте реализовать свой способ на практике.
7. Можно ли внести изменения в текст, полученный при помощи сканера? (Указание: постарайтесь проанализировать как можно больше вариантов.)
8. Подумайте, каким образом в матричный принтер мог вводиться новый шрифт?
9. Есть ли отличия между печатью текста и выводом рисунка? Одинакова ли в связи с этим скорость их печати?
10. Способен ли ваш монитор работать в чисто текстовом режиме? Видели ли вы когда-нибудь старые программы, которые это делают?
11. Каким образом процессор может адресоваться к устройствам ввода/вывода? Что такое порт ввода/вывода?
12. Опишите примерный ход взаимодействия между процессором и принтером. Как вы считаете, что происходит раньше: очередной символ печатается на бумагу или появляется сигнал готовности к приему следующего?
13. Что такое прерывание и как оно работает? Проанализируйте приведенные в тексте аналогии и попробуйте придумать собственные.
14. Имеют ли место прерывания в следующих ситуациях и почему: а) в программе встречается оператор ввода и машина принимает число; б) во время вычислений пользователь нажимает клавишу <F>, режим отображения информации на экране изменяется, а счет продолжается.

6.7. Любопытные эксперименты

6.7.1. Как отображаются вводимые символы?

Проиллюстрируем положение, высказанное в *разд. 6.1.1* о том, что ввод с клавиатуры и вывод на экран монитора — это разные, хотя и связанные между собой процессы.

Цель эксперимента. Подтвердить, что выводимое на экран так называемое эхо от ввода с клавиатуры существенным образом определяется программным обеспечением.

Для проведения эксперимента используем несложную программу на языке Basic, приведенную в листинге 6.1.

Комментарий к выбору языка

Программы для предыдущих экспериментов были написаны на языке Паскаль. Почему эта реализована на другом языке? Причина чисто техническая. Для составления эквивалентной программы на языке Паскаль вместо применяемой функции INKEY$ потребовалось бы использовать ее аналог READKEY, описанный в стандартном модуле CRT. В силу досадного несоответствия использование этого модуля языка Турбо Паскаль на машинах с процессорами тактовой частоты более 200 МГц может вызывать проблемы. Чтобы их избежать и не заставлять читателей вникать в тонкости, автор принял решение использовать для данного эксперимента язык Basic. В очередной раз подчеркнем, что результаты описываемых в книге экспериментов не связаны с выбором того или иного языка программирования.

Листинг 6.1. Программа для проверки отображения вводимых символов

```
DO: s$ = INKEY$: LOOP WHILE s$ = ""
IF s$ = "P" THEN
   PRINT "Pentium ";
   DO: s$ = INKEY$: LOOP WHILE s$ = ""
   IF s$ = "2" THEN PRINT "II";
   IF s$ = "3" THEN PRINT "III";
   IF s$ = "4" THEN PRINT "IV";
END IF: PRINT
END
```

Данная программа умеет по двум введенным символом распознавать и выводить на экран полное название некоторых модификаций процессора Pentium: так, при нажатии заглавной латинской буквы "P" и цифры "2" она выводит "Pentium II". Аналогично обрабатываются комбинации с цифрами "3" и "4". Несоответствующий описанным правилам ввод просто игнорируется, и на экране ничего не появляется.

Рассмотрим приведенную выше программу чуть подробнее. В первой строке написан цикл ожидания ввода символа: в переменную S$ с помощью функции INKEY$ считывается содержимое буфера клавиатуры. Цикл повторяется до тех пор, пока буфер пуст, т. е. не нажата ни одна клавиша. Для случая, когда вводится символ "P", выполняется ввод второго символа, который анализируется на совпадение с цифрами "2" — "4"; при совпадении выводится имитация "римского эквивалента" введенной цифры, в противном случает на экране ничего не появляется. Если же первый символ был отличен от "P", то все описанные выше действия игнорируются. В любом случае программа завершается переводом строки на экране с помощью оператора PRINT.

Вывод. Эксперимент убедительно показывает, что ввод с клавиатуры и его эхо на экране не являются жестко связанными. Мы видели, что эхопечать символов определяется тем, как их обрабатывает программное обеспечение.

6.7.2.* Реализация печати символов на низком уровне

В *разд. 6.2.2* лекции описан возможный сценарий диалога процессора с принтером при выводе символа. Проведем эксперимент, позволяющий посмотреть, как в действительности на самом низком уровне происходит взаимодействие машины с печатающим устройством. Подчеркнем, что данный эксперимент является достаточно тонким: в принципе его результаты могут зависеть от типа принтера и версии операционной системы. Вообще говоря, гарантированный успех определяется принципом "чем старее — тем лучше": идеально к описанному эксперименту подходит матричный принтер в среде MS-DOS. Тем не менее, все не так мрачно и данные о нескольких конфигурациях, в которых опыт прошел удачно, приведены в конце раздела. Если на вашем компьютере эксперимент все же не удался, придется ограничиться чтением теории.

Цель эксперимента. Изучить диалог между процессором и печатающим устройством в ходе вывода строки символов на бумагу.

Как обычно, к сложному эксперименту необходимо провести основательную предварительную теоретическую подготовку.

Наш эксперимент во многом опирается на книгу [18] (страницы указаны в комментариях к листингу 6.2). Книга является прекрасным сборником приемов доступа к устройствам компьютера IBM PC, причем на трех разных уровнях: высоком (в цитированной книге используется язык Basic), среднем (через функции MS-DOS; аналог нашего эксперимента, описанного в *разд. 5.6.1*) и низком (непосредственно через регистры процессора и порты); нам сейчас нужен именно последний. К сожалению, книга уже морально устарела, но пока еще ее программы продолжают работать в среде Windows, в чем мы и собираемся убедиться.

Примечание

Читатели вправе спросить: а почему мы все время пишем программы в безнадежно устаревшей среде MS-DOS, а не напрямую в Windows? Надо четко представлять, что в более примитивных операционных системах гораздо проще добраться до оборудования, чем в более сложных. Иначе говоря, попытки в Windows вывести байт непосредственно в какой-либо порт могут просто пресекаться. Примером могут служить DOS-версии программ из пакета Norton Utilities: поскольку они построены на доступе к диску на низком уровне, в среде Windows они просто не работают. Гораздо менее существенной причиной (но все же не стоит ее сбрасывать со счета) является большое количество хорошей литературы по "старой доброй" MS-DOS. В самом деле, вы видели доступную книгу по программированию доступа к портам принтера из Windows?

Итак, начнем с деталей управления принтером. Он имеет три порта с последовательными номерами (табл. 6.1).

Таблица 6.1. *Порты принтера в IBM PC и их номера*

Порт	Номер
Порт выходных данных	N
Порт статуса	N+1
Порт управления	N+2

Порт входных данных служит для передачи символа принтеру. Порт состояния сообщает разнообразную информацию о принтере, например, готов ли он к работе и не было ли ошибки при выводе предыдущих символов. Порт управления инициализирует принтер и управляет выводом данных. Начальный адрес портов для выхода LPT1, обозначенный в таблице `N`, хранится в специальной таблице MS-DOS и имеет адрес 0040:0008. Данный адрес указан сегментным способом *(см. разд. 4.3.2)*, характерным для MS-DOS, и состоит из сегмента 0040 и смещения 0008 в этом сегменте.

В программе также будут использоваться отдельные биты портов. В частности, для порта статуса мы будем анализировать биты 3-й и 7-й (общепринятым способом является нумерация от нуля начиная с самого младшего бита). Бит 3 устанавливается в единицу в случае ошибки, а бит 7 несет информацию о готовности принтера: 1 означает, что принтер свободен. Значения остальных битов этого порта для нашего эксперимента несущественно.

Для выделения указанных выше битов требуются шестнадцатеричные константы 8 и 80 соответственно. Напомним читателям, что вопросы применения логических операций для выделения отдельных разрядов кода рассматривались в *разд. 3.6.4.*

В регистре управления для нас наиболее важным будет младший, 0-й бит, который соответствует стробирующему сигналу. Кроме того, мы будем задавать и еще несколько битов этого регистра, влияющих, например, на выбор принтера и запрет прерываний от него. Но поскольку непосредственного влияния на изучаемый диалог они не оказывают, предлагается их нормальное состояние специально не обсуждать.

Перейдем к анализу программы для эксперимента, приведенной в листинге 6.2.

Листинг 6.2. Программа, реализующая печать символов на низком уровне

```
PROGRAM print_text;
{По книге: Р.Джордейн. "Справочник программиста
персональных компьютеров типа IBM PC и XT", с.415-417}
VAR   s:string;  {печатаемая строка}
      k:INTEGER; {длина строки}
```

```
BEGIN s:='Print test'+chr(10)+chr(13)+chr(12);
      k:=length(s);
      ASM mov ax,40h     {адрес порта принтера}
          mov es,ax      {  узнать у MS-DOS и}
          mov dx,es:[8] {  поместить в dx}
          lea bx,s+1     {адрес текста в bx}
          mov cx,k       {количество символов в cx}
      @1: mov al,[bx]    {очередной символ в al}
          out dx,al      {вывести в порт данных}
          inc dx         {получить в dx номер}
          inc dx         {  порта управления выводом}
          mov al,13      {биты для установки строба}
          out dx,al      {посылаем сигнал строба}
          dec al         {убираем бит строба}
          out dx,al      {выводим: убрали строб}
          dec dx         {dx на порт статуса}
      @2: in  al,dx      {прочитать порт статуса}
          test al,8      {есть ошибка?}
          nop            {здесь выход при ошибке}
          test al,80h    {принтер занят?}
          jz @2          {  повторять, если да}
          inc bx         {адрес следующего символа}
          dec dx         {dx на регистр данных}
          loop @1        {если cx<>0, продолжать}
      END;
      WRITELN('Print is done')
END.
```

В строковую переменную `s` заносится текст для печати, дополненный кодами конца строки (10 и 13), а также переводом страницы (12). Все эти управляющие коды используются для того, чтобы система имела основания произвести печать, не дожидаясь, пока заполнится весь будущий лист. После этого в переменную `k` заносится полная длина сформированной строки.

Дальше следует вставка на Ассемблере, которая и обеспечивает печать. Она начинается с загрузки в регистры необходимых значений. В регистр `ES` заносится значение сегмента 0040 и байт со смещением 8 считывается в регистр `DX`. Если вы помните, по адресу 0040:0008 находится начальный адрес портов принтера. В `BX` подготавливается адрес текста, который на 1 больше, чем адрес начала переменной `s` требуется пропустить нулевой байт строковой переменной, в котором хранится ее длина. Наконец, последнее значение, равное количеству символов, заносится в качестве счетчика циклов в регистр `CX`.

Далее после метки `@1` начинается цикл. Очередной символ через `AL` выводится в порт данных. Затем адрес в `DX` увеличивается на 2, "перестраиваясь"

на порт управления, куда заносится код 13. Для нас наиболее важно, что в нем установлен самый младший бит, т. е. это действие приводит к установке строба. По нему принтер понимает, что процессор подготовил данные. После этого из AL вычитается 1, следовательно, бит строба сбрасывается в ноль, и полученная константа снова выводится в порт.

После этого начинается следующий этап обмена. Адрес в DX перестраивается на порт статуса и считывается значение последнего. Далее следует его анализ: проверяется бит, несущий информацию об ошибке (в нашей программе выход по ошибке не предусмотрен, поэтому стоит команда nop — нет операции), и бит готовности принтера. При неготовности цикл повторяется, начиная с метки @2; в противном случае вычисляется адрес следующего символа текста в BX, а DX устанавливается на порт данных. Последним действием цикла команда loop вычитает 1 из CX и сравнивает его результат с нулем, иначе говоря, проверяет, остались ли еще ненапечатанные символы; если да — следует переход на метку @1 и повторяется вывод следующего символа.

После завершения работы ассемблерной части программа выводит на дисплей диагностический текст, подтверждая успешное окончание цикла.

Таким образом, мы очень подробно проследили за диалогом между процессором и принтером. Остается проверить, что вся эта теория работает. Как уже отмечалось выше, работа на низком уровне через порты — это тонкий эксперимент, который может не получиться. Поэтому автор провел несколько экспериментов с различными машинами, принтерами и операционными системами (табл. 6.2).

Таблица 6.2. *Список ЭВМ, операционных систем и принтеров для тестирования печати на низком уровне*

Процессор	ОС	Принтер
Celeron, 333 МГц	Windows 98	Epson LX-300 (матричный)
Pentium III, 1100 МГц	Windows 2000	Epson LX-1050+ (матричный)
Celeron, 600 МГц	Windows 98	Epson Stylus 200 (струйный)
Pentium MMIX, 233 МГц	Windows 98	HP LaserJet 6L (лазерный)

Все они были удачными. Надеюсь, у читателей принтер также не окажется каким-нибудь нестандартным, так что "проигнорирует" предложенный ему диалог.

Вывод. Описанный выше диалог процессора и принтера действительно приводит к печати на бумагу требуемой строки.

6.7.3. Прямая запись в видеопамять

Некоторым специфическим методом вывода видеоинформации на экран монитора является ее копирование напрямую в видеопамять. Поскольку контроллер дисплея непрерывно отображает ее содержимое на экран, то для изменения изображения достаточно просто поменять значения необходимых байтов видеопамяти. Хотя данный метод вывода является простым и очень быстрым, его использование без особой необходимости не рекомендуется, поскольку организация видеоОЗУ не обязательно является стандартной и по мере развития схем видеоадаптеров в будущем может измениться.

Цель эксперимента. Проверить, что если имеется область памяти, содержимое которой непрерывно отображается на экран монитора, то это свойство можно использовать для организации вывода информации.

Практическое применение рассматриваемого метода требует четкого знания принципов планирования видеопамяти. Поэтому обсуждение вопроса целесообразно начать с размещения информации в видеопамяти. Естественно, мы выберем для подробного ознакомления наиболее простой текстовой режим. Необходимые сведения по его устройству были в основном взяты из книги [1].

Каждому выводимому на текстовой экран символу в памяти IBM PC отводится 2 байта: в первый заносится собственно код символа, а во второй — его атрибут, описывающий цветовые характеристики (младшие 4 бита определяют цвет самого символа, а в старших задается цвет фона и режим мигания) (рис. 6.1).

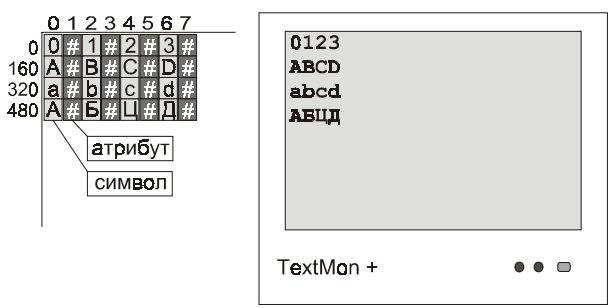


Рис. 6.1. Устройство видеопамяти IBM PC в текстовом режиме

Поскольку стандартная строка дисплея состоит из 80 символов, то самая верхняя строка экрана занимает байты с адресами 0—159, следующая 160—319 и т. д. Символы строк следуют непосредственно друг за другом, поэтому при занесении длинного текста переход после заполнения очередной строчки к новой происходит автоматически. Для завершения картины остается

добавить, что начальный шестнадцатеричный адрес видеопамяти равен B8000.

Перейдем теперь к программной реализации эксперимента, приведенной в листинге 6.3.

Листинг 6.3. Программа прямой записи в видеопамять

```
PROGRAM videoRAM;
CONST str1=2;     {номер первой строки с текстом}
      sym=32;     {первый символ}
VAR   s,o:WORD;   {адрес: сегмент и смещение}
      y:INTEGER; {смещение по Y}
BEGIN s:=$B800;   {начало видеопамяти}
      y:=160*(str1-1);
      FOR o:=0 TO 209 DO
          BEGIN mem[s:y+2*o]:=sym+o; {символ}
                mem[s:y+2*o+1]:=o;   {атрибут}
          END
END.
```

Константа `str1` задает номер строки экрана, начиная с которой будут выводиться символы; по ней вычисляется смещение, обозначенное в программе `y`, которое добавляется ко всем адресам видеопамяти. А коды выводимых символов начинаются с 32 и будут циклически увеличиваться на единицу, последовательно "пробегая" весь алфавит машины.

Программа состоит из единственного цикла, который выводит 210 символов непосредственно в видеопамять. Последовательные коды символов помещаются в байты с четными номерами, а соответствующие им атрибуты, также возрастающие на единицу — в нечетные адреса.

Если теперь запустить нашу программу, то вторая, третья и часть четвертой строки экрана заполнятся разноцветными символами, причем фон также будет иметь периодически изменяемые цвета. Поскольку воспроизвести эту картинку на страницах книги затруднительно, ограничимся данным выше словесным описанием результата.

Завершая изложение эксперимента, автор вынужден честно написать, что наблюдаемый нами на экране эффект на самом деле заметно сложнее, чем кажется с первого взгляда. Только в среде MS-DOS указанные адреса реально соответствуют видеопамяти — в Windows же поведение видеопамяти DOS эмулируется программно. Тем не менее, механизм работает, что подтверждает правильность наших общих представлений.

Вывод. Прямая запись данных в область памяти, которая отображается на экране, эквивалентна выводу на дисплей.

6.7.4. Изучение событий, связанных с мышью

Обсуждая в лекции принципы обслуживания внешних устройств, мы говорили о прерываниях и о механизме событий, который используется в современных системах программирования для их обработки. В данном эксперименте попробуем понаблюдать за простейшими событиями, связанными с работой мыши.

Цель эксперимента. Наблюдение за событиями, порождаемыми мышью.

Для реализации эксперимента нам потребуется более современная программная среда, чем мы использовали раньше, например, Delphi или Visual Basic. В операционной системе Linux можно использовать среду разработки Kylix, внешне полностью аналогичную Delphi. Ниже будет описываться первая из систем, работа с другими выглядит очень похоже. Поскольку в задачи книги совсем не входит описание основ программирования в Delphi, мы ограничимся простейшей демонстрацией.

Итак, запустим систему Delphi любой доступной вам версии. Внимательно осмотрев экран, найдем в его верхней части главное окно системы. Его важной частью является так называемая палитра компонентов, на которой изображены имеющиеся в распоряжении системы стандартные элементы для наших будущих программ. Нас сейчас будет интересовать единственный компонент, который называется **Label** (текстовая метка). Найти его вам поможет рис. 6.2, на котором указатель мыши подведен к интересующему нас компоненту. Надпись с названием компонента не подрисована дополнительно, а есть стандартная реакция системы, которую вы тоже увидите, если некоторое время подержите указатель мыши неподвижно.

Теперь поместим метку на заготовку будущего окна **Form1**. Это делается совсем просто — щелкнем левой кнопкой мыши по компоненту, поместив указатель в любое понравившееся место формы щелкнем еще раз, показывая системе выбранное место. Если вы все сделали правильно, на форме появится выделенный текст **Label1**. Обязательно снимите выделение, для чего щелкните мышью по любому пустому месту на форме.

Метка, на которую будет выводиться индицируемый текст, готова. Теперь остается написать обработчик событий мыши. Для этого обратим свое внимание на другое служебное окно, которое называется **Object Inspector** (Инспектор объектов). На рис. 6.2 оно находится в левой части, а его более крупный вид изображен на рис. 6.3.

Обязательно проверьте, что Инспектор объектов настроен на работу с формой, о чем говорит надпись **Form1** в верхней части окна. Если вы после создания метки не забыли щелкнуть по форме, это требование уже выполнено. Теперь выберем вкладку **Events** — события (см. указатель мыши на рис. 6.3). Найдем в перечне событий Инспектора интересующее нас сейчас движение мыши, которое в Delphi называется **OnMouseMove**. Дважды щелкнем по

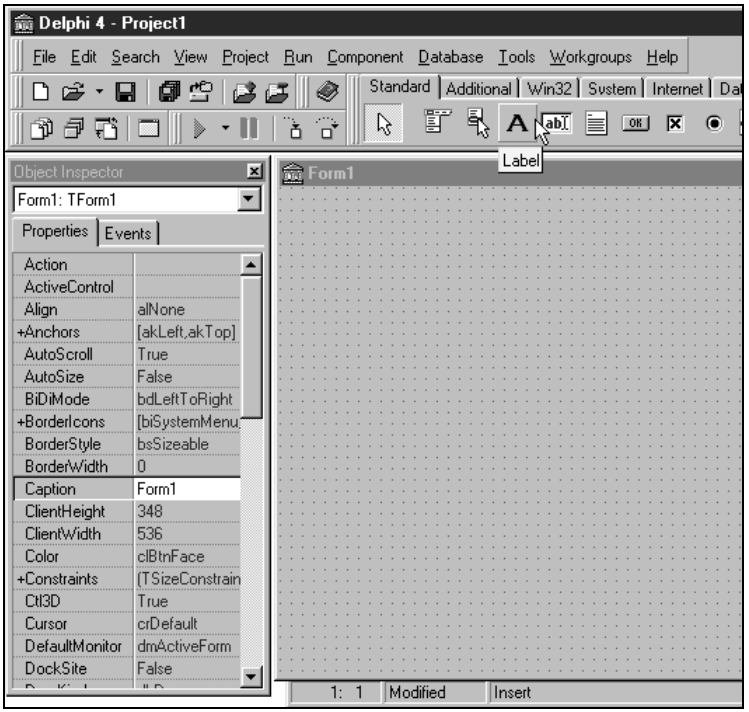


Рис. 6.2. Вид системы Delphi и выбор компонента **Label**

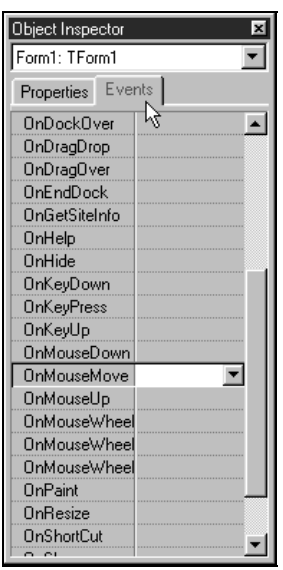


Рис. 6.3. Инспектор объектов и выбор события **OnMouseMove**

правому столбику таблицы в данной строке и с удивлением обнаружим, что Delphi перешла в окно с текстом программы. Ни в коем случае не двигайте курсор — система уже установила его именно туда, куда требуется! Просто очень аккуратно наберите следующую строку:

```
label1.caption := intToStr(X) + ', ' + intToStr(Y)
```

Эта строка будет при каждом перемещении мыши брать ее координаты `X` и `Y`, преобразовывать их, пользуясь функцией `intToStr()` из числовой формы (`integer`) в строковую (`string`) и через запятую помещать в отображаемую область метки **Label1**, обозначаемую в Delphi свойством **Caption**. Как ни странно, это уже все. Остается только запустить полученное приложение и провести собственно эксперимент.

Для запуска программы достаточно просто нажать клавишу <F9>. Проделаем это и понаблюдаем, как приложение отслеживает перемещение мыши (рис. 6.4).

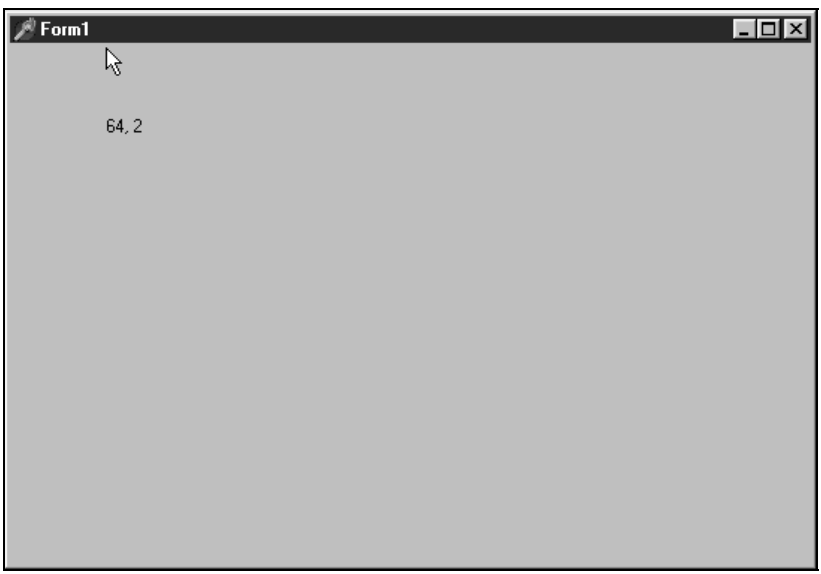


Рис. 6.4. Работа приложения по отслеживанию перемещения мыши

Обратите внимание, что указатель мыши должен обязательно двигаться в пределах площади окна, иначе его связь с нашим приложением теряется!

Если вам понравился эксперимент, вы легко его можете продолжить. Для этого обязательно закройте окно с приложением, следящим за мышью (остановите его работу), и обратитесь к тексту программы. Если по какой-то причине он уже не виден, снова воспользуйтесь Инспектором объектов, как это было подробно описано выше. Аккуратно сотрите тот текст, который вы

так старательно набирали и оставьте вместо него пустую строку (все остальное система уберет сама). Пользуясь уже описанными приемами написания обработчика на событие, соответствующее нажатию кнопки мыши **OnMouseDown**, добавьте строку

```
label1.caption := 'Mouse button down'
```

а на отпускание кнопки — **OnMouseUp**

```
label1.caption := 'Mouse button up'
```

Снова запустите программу и понаблюдайте, как она будет реагировать на нажатие и отпускание клавиши мыши.

На прилагаемом к книге CD-диске приведена программа первого из описанных экспериментов. Дополнительно в проект добавлен таймер, который еще на одну метку выводит свои непрерывно возрастающие показания. Сделано это с целью демонстрации того, что машина успевает не только следить за работой таймера, но и прерывается на обработку движения мыши — типичный образец многозадачной работы.

> **Вывод.** Современные программные системы, используя принцип обработки событий, позволяют легко обрабатывать прерывания от внешних устройств, например мыши.

6.7.5. Поиск пикселов на мониторе

Идея и описание этого любопытного эксперимента изложены в газете "Информатика" [43]. Предлагается, следуя рекомендациям некоторых учебников, вооружиться увеличительным стеклом и попытаться увидеть триады красных, зеленых и синих (RGB — от англ. "Red-Green-Blue") точек на экране монитора.

> **Цель эксперимента.** "Встанем же на место особо любопытного ученика, который, вооружившись лупой, попытается рассмотреть пикселы на экране традиционного монитора или телевизора, сделанных на основе электронно-лучевой трубки" [43].

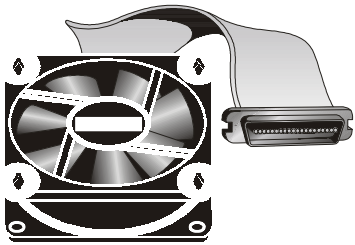
Однако, как предупреждает нас первоисточник, результаты экспериментов будут успешными далеко не всегда. Причина в том, что существуют разные технологии изготовления электронно-лучевых трубок, причем только одна из них (так называемая "теневая маска") позволит увидеть настоящую мозаику точек. В других случаях, когда вместо маски с отверстиями используется система нитей из люминофора трех основных цветов (апертурная решетка), картина будет совсем другой. В цитируемой статье [43] приведены очень наглядные фотографии трех типичных картин, которые могут увидеть "любопытные ученики".

К материалу данной замечательной статьи сто́ит лишь добавить, что желательно различать понятия "точки экрана", являющиеся физическими реаль-

но существующими объектами, и пикселы, которые следует считать логическими элементами изображения. Вспомним, что существует несколько типичных конфигураций картинки на экране монитора IBM PC: 640 × 480 пикселов, 600 × 800 и некоторые другие. Но на одном и том же мониторе можно установить любую из них! Это значит, что пикселы не тождественны точкам монитора, и каждый из них может быть образован несколькими соседними светящимися точками (в пределе — одной). Когда мы просим машину сделать тот или иной пиксел, например, синим, то программное обеспечение, учитывая установленный режим дисплея, закрасит одну или несколько соседних точек монитора.

Вывод. Наблюдать точки изображения "в явном виде" можно не на всех мониторах: картина, которую мы увидим под увеличительным стеклом, зависит от устройства электронно-лучевой трубки. Еще раз подчеркнем, что не следует отождествлять логическое понятие "пиксел" с "физической точкой" экрана.

ЛЕКЦИЯ 7



Роль программного обеспечения

Наверно, у некоторых читателей, внимательно изучивших предыдущие лекции, возник вопрос: "Как же так? Получается, что современный компьютер построен на относительно простых и понятных принципах, и нет в нем ничего архисложного и таинственного. Тогда каким образом, имея столь ограниченную систему операций, он может распознавать тексты и речь, оживлять объемные рисунки, играть в шахматы и делать прочие разнообразные и впечатляющие вещи?" Все дело в том, что "интеллектуальным" компьютер делает не только (и, может быть, даже не столько) огромное быстродействие и гигантская память, но и еще один важный компонент — ***программное обеспечение***, содержащее подробнейшие инструкции по решению тех или иных практических задач. Подобно новорожденному ребенку (каким бы талантливым он не был), компьютер также абсолютно беспомощен до тех пор, пока внутрь него не попадут формализованные человеком знания, как действовать в самых разнообразных ситуациях.

Роль программного обеспечения в работе компьютера и будет темой нашей лекции.

7.1. Компьютер — единство аппаратной и программной частей

Современный компьютер действительно является воплощением подлинного единства программной и аппаратной частей. Не только написание программ без машины лишено практического смысла, но и компьютер без программного обеспечения не более чем дорогое украшение стола.

Значение *аппаратной* (hardware) и *программной* (software) частей для современного компьютера трудно переоценить. О роли интересующей нас сейчас программной части говорит хотя бы тот факт, что стоимость современных программных средств, по крайней мере, не меньше стоимости самой маши-

ны, включая все внешние устройства. "Сегодня никого не удивляет, когда стоимость установленного программного обеспечения больше, чем стоимость самого компьютера. ...Маленькая фирма с 5-ю компьютерами потратит 7—10 тыс. долларов только на программное обеспечение. Но это только начало. ...За всю жизнь компьютера (3—5 лет)... на программное обеспечение придется потратить порядка 2—5 тыс. долларов". [75]

В нашей стране в силу исторически сложившихся "национальных особенностей" данный тезис не совсем очевиден и, возможно, требует доказательств. Рассмотрим их подробнее.

Возьмем к примеру таблицу изменения относительной стоимости программных и аппаратных средств (табл. 7.1), построенную по данным [69]. Из нее отчетливо видно, что роль программных средств в работе вычислительной техники неуклонно возрастала и по мере ее развития стала очень существенной.

Таблица 7.1. *Соотношение стоимости программной и аппаратной частей компьютера*

Относительная стоимость	50-е	60-е	70-е	80-е
Аппаратные средства	70 %	35 %	20 %	15 %
Программные средства	30 %	65 %	80 %	85 %

Примечание

В момент введения в школу курса основ информатики и вычислительной техники (напомним читателям, что процесс имеет четкую дату начала — 1985 год) подобных таблиц, анализировавших те или иные тенденции развития ВТ, в популярных книгах и журналах публиковалось много. Сейчас в учебной литературе попыток анализа стало заметно меньше, в чем также видится переход от фундаментальных принципов к прагматическим.

К сожалению, более поздних данных для продолжения таблицы при подготовке лекции мне найти не удалось. Поэтому давайте проделаем некоторую оценку сами.

Для начала обратимся к "первоисточникам", т. е. посетим интернет-сайты известных фирм-производителей программного обеспечения. Например, визит на официальный сайт компании Corel Corporation позволил легко установить, что знаменитый векторный графический редактор CorelDRAW последней (на момент написания этих строк — март 2003 года) версии 11 продавался за $399. Не менее известный растровый редактор Photoshop 7, который в мире считается одним из самых популярных пакетов для редактирования фотографических изображений, разработавшей его фирмой Adobe, был оценен в $607. Из других предлагаемых на этом же сайте продуктов

стоит упомянуть профессиональный видеоредактор Adobe Premier 6.5, предлагавшийся посетителям за $399. Подчеркнем, что приведенные цены можно считать минимально возможными, поскольку в них не включены никакие накладные расходы, в частности стоимость доставки. Если посмотреть прайс-листы отечественных компьютерных фирм, то мы это немедленно заметим.

Интересно рассказать о результатах посещения сайта компании Borland, где есть специальный online-магазин для продажи программных разработок фирмы. Одна из лучших систем программирования Delphi 7 Personal приятно порадовала своей "дешевизной" — "всего" $99 (правда, стоит подчеркнуть, что она предназначена исключительно для некоммерческого написания программ!). Зато профессиональный вариант для производителей программного обеспечения Delphi 7 Studio Enterprise Edition стоил около $3000 (рис. 7.1). Аналогичная разница четко прослеживалась и для остальных программных продуктов — дешевые "ознакомительные" версии и дорогие инструменты реальных коммерческих приложений. Отметим, что, как и другие производители ПО, Borland предоставляет очень существенные скидки образовательным учреждениям.

Рис. 7.1. Типичный лицензионный комплект программного продукта: компакт-диск, документация и "фирменная" упаковочная коробка

К сожалению, попытка найти какие-либо сведения о стоимости (хотя бы примерной) программных продуктов Microsoft на сайте компании оказалась безуспешной: гигант "софтверной" индустрии отправил за информацией к своим дилерам. Причем по указанной на сайте ссылке подобной инфор-

мации тоже не оказалось. Тем не менее, с точки зрения нашей оценки стоимость продуктов этой фирмы как раз очень важна. Чтобы как-то выйти из положения, пришлось анализировать прайсы отечественных фирм, предлагавших эти продукты своим клиентам. Как и следовало ожидать, картина получилась довольно пестрой. Чтобы не запутаться, за основу была взята стоимость полной, так называемой "коробочной", версии продукта на русском языке (кстати, любопытно, что английские версии всегда дороже!).

Примечание

Существует особый способ продажи ПО вместе с новыми компьютерами. Такие версии заметно дешевле, но когда наблюдаешь на практике реализацию этого процесса в некоторых наших торгующих компьютерами фирмах, то возникают серьезные сомнения по поводу законности происходящей процедуры. Поэтому данная разновидность цены здесь не обсуждается.

Какими же оказались примерные цены на продукты Microsoft? Наиболее дешевые предложения русской версии операционной системы Windows Pro 2000 на компакт-дисках колебались около отметки $250. Версия Windows XP Home Edition (т. е. вариант для домашнего пользования) стоила, естественно, меньше — в районе $170. Зато существенно более высокие цены были установлены на Microsoft Office: его текущая стоимость для последней версии XP была около $400.

Примечание

Справедливости ради следует сказать, что существует абсолютно бесплатное программное обеспечение. Но следует ли из этого вывод, что в него было вложено меньшее количество сил и времени программистов, чем в разработку коммерческих версий аналогичного ПО?

Мне не хотелось бы сейчас вдаваться в детали и обсуждать уровень приведенных выше в качестве примера цен, их обоснованность и существующие возможности уменьшения. Моя цель, как я уже говорил, значительно скромнее: показать читателям, что реальная стоимость типичного программного обеспечения, по крайней мере, никак не меньше стоимости самого компьютерного оборудования (обычно принято считать, что хорошо оборудованный компьютер по порядку величины обойдется в $1000). Поэтому не стоит спорить, что важнее — технические характеристики машины или возможности установленных на ней программ; существенное значение имеет и то, и другое, а еще точнее — гармония обеих составляющих.

Официальное приобретение программного обеспечения, о котором шла речь выше, является в цивилизованном мире общепринятым. Подобное ПО, называемое часто *лицензионным*, дает его обладателю такие преимущества, как:

- гарантии качества и стабильной работы (отсутствие сбоев носителя, завершенность версии, полный комплект необходимых компонентов, работоспособность во всех режимах и т. д.);

- техническая поддержка по любым вопросам, связанным с установкой и эксплуатацией продукта;
- подробная сопроводительная документация (например, для изображенной на рис. 7.1 среды программирования, в комплект входит 5 книг, включая 450-страничное руководство для начинающего пользователя; к Турбо Паскалю прилагается более десятка книг);
- информация о новых продуктах и обновление текущих версий с большой скидкой;
- соответствие закону (что для организаций теоретически может быть проверено в любой момент).

Примечание

К сожалению, большая часть документации для иностранных программных продуктов обычно написана на английском языке. А вы сами бы стали переводить книги на язык той страны, где их легально приобретает лишь незначительная часть населения?

Тотальное распространение пиратского программного обеспечения в нашей стране сильно размывает рассмотренную картину, но это отдельный разговор, выходящий за рамки наших лекций по основам устройства вычислительной техники. Отмечу только одну деталь. Отнюдь не пытаясь показать себя самым праведным человеком, все же хотел бы напомнить читателям, что не стоит гордиться и хвастаться всем подряд установкой самых последних пиратских версий ПО и тем более искать обоснование "преимуществ" подобного способа распространения программного обеспечения. Как ни удивительно, мне приходилось встречать людей, которые искренне верили, что раз они приобрели "за свои кровные" диск с программами в центральном универмаге, то установленную на их компьютере копию программного продукта уже можно называть законной. Особенно сказанное выше актуально для школьных учителей, которые при достаточном оптимизме имеют некоторые основания надеяться, что их ученики смогут жить при более цивилизованной организации жизни[1].

Примечание

Информация из журнала [80] без комментариев. В настоящее время Министерство юстиции США рассматривает вопрос о начале активного использования в судебной практике Акта против электронных хищений (No Electronic Theft Act, сокращенно NET Act), подписанного еще в 1997 году. По этому закону за решетку можно попасть не только за использование пиратского программного обеспечения, но и просто за загрузку из Интернета MP3-файлов.

---

[1] Между прочим, а не является ли одной из причин, по которой сошел на нет популярный некогда лозунг о программировании как о второй грамотности, то, что результаты труда в этой области все равно не будут по достоинству оценены и оплачены? А вот Индия, например, занимает по экспорту ПО одно из первых мест в мире и имеет от этого большие доходы.

В заключение раздела подчеркну одну важную мысль относительно роли программного обеспечения в работе современных компьютеров. Наличие ПО позволяет приспособить одно и то же компьютерное оборудование к потребностям специалистов из самых разнообразных областей. В результате и писатель, и бухгалтер, и инженер-технолог пользуются абсолютно одинаковыми вычислительными машинами. Таким образом, здесь вновь работает та же самая идея, что и при производстве микропроцессоров: универсальные изделия, выпускаемые в больших количествах, получаются существенно дешевле (вспомните подробности, изложенные в *разд. 3.1.1*).

7.2. Типы программного обеспечения

Существуют различные способы классификации программного обеспечения. Мы рассмотрим простейший из них [70, 34], согласно которому все ПО делится на три категории: *системное*, *прикладное* и *системы программирования*. И хотя с точки зрения нашего курса интерес представляет в основном первая группа, для полноты картины рассмотрим все три.

7.2.1. Системное ПО

Для работы компьютера, прежде всего, требуется *системное программное обеспечение*, которое служит основой нормального функционирования компьютера и согласованной работы всех его узлов. Эта "техническая" (вспомогательная) часть ПО хотя и не решает конкретных задач пользователей, но зато создает возможности для их решения. К данной категории программного обеспечения принадлежит в первую очередь операционная система, которая будет подробнее рассмотрена в *разд. 7.4*, а также всевозможные тестирующие и сервисные служебные программы (часто их еще называют *утилитами*). Сюда же относятся программы форматирования, проверки и дефрагментации дисков, архиваторы, антивирусные и некоторые сетевые программы.

При покупке нового компьютера минимальный набор системного программного обеспечения, как правило, сразу же устанавливается, хотя бы ради того, чтобы показать покупателю, что компьютер работает.

7.2.2. Прикладное ПО

Прикладное программное обеспечение, как следует из названия, предназначено для решения прикладных задач, "т. е. конкретных задач производственного, научного, управленческого или учебно-тренировочного характера" [33].

Среди этой категории выделяется, прежде всего, несколько программ наиболее общего назначения: текстовые и графические редакторы, электронные

таблицы, системы управления базами данных и программы для межмашинных сетевых коммуникаций. Их особенностью является то, что они могут использоваться в работе на компьютере практически во всех областях.

Более специализированное программное обеспечение принято называть пакетами прикладных программ. К ПО такого типа относятся, например, различные математические пакеты (Maple, Mathcad), бухгалтерские (1С, Бест) или многочисленные пакеты обучающих программ.

Именно ради этой группы программного обеспечения в основном и приобретает компьютеры большая часть пользователей.

7.2.3. Системы программирования

Хотя многие авторы относят средства программирования к системному ПО, все же лучше выделить их в отдельную категорию. Ее отличительной чертой является создание новых программ для ЭВМ. Благодаря системам программирования любой "компьютерно грамотный" человек может сам научить машину делать то, что та раньше не умела, благодаря чему существенно снижается его зависимость от необходимости поиска написанных кем-то программ.

Уточним, что термин "система программирования", сменивший более ранний "язык программирования", был введен потому, что в комплект современного транслятора обязательно входит целый набор дополнительных программных средств, которые облегчают работу программиста: специализированные текстовые редакторы, способные выделять служебные слова и располагать их друг под другом определенным образом; различные отладочные средства, помогающие найти ошибку в "упорно не желающей работать" программе; средства компоновки с модулями из других языков и т. д.

7.3. "Слои" программного обеспечения

Современное программное обеспечение имеет "слоистую" структуру, при которой каждый слой активно использует возможности слоев более низкого уровня. Как образно сказал М. Отставнов [66], "то, что пользователю, сидящему за монитором, представляется сплошной графической операционной средой, реализовано как многослойный сандвич технологий". Рассмотрим, как выглядят эти слои в самом общем виде, обратившись к следующей схеме (рис. 7.2).

Итак, пользователь главным образом работает с прикладными программами. Почти все современные пакеты написаны с помощью систем программирования, и поэтому в значительной степени опираются на общие блоки (лучше сказать, библиотеки) языков программирования. Все это активно использует операционную систему, которая, в свою очередь, также состоит из

нескольких взаимодействующих слоев: программы общения с пользователем, аппаратно-независимого ядра системы и слоя, обеспечивающего взаимодействие с конкретными периферийными устройствами. Лишь самый "нижний" из этих программных слоев непосредственно управляет работой аппаратной части компьютера.

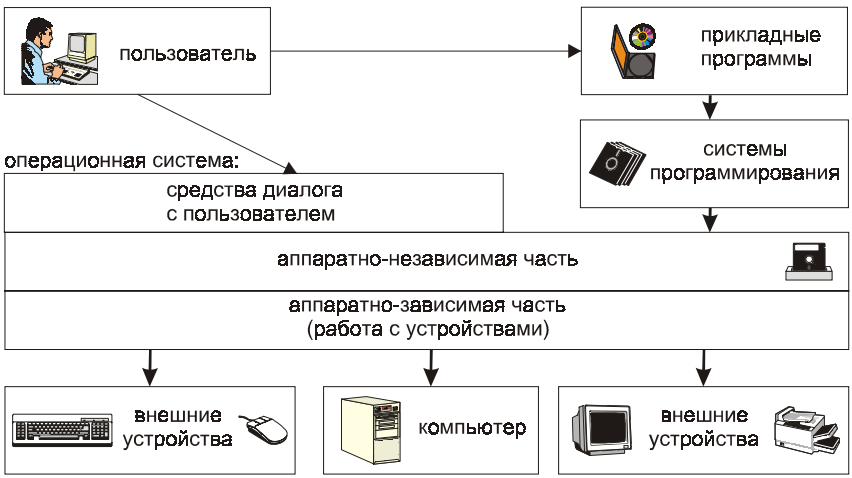


Рис. 7.2. "Слои" программного обеспечения

Дополняя описанную картину, отметим, что запуск прикладных программ является одной из многочисленных функций программных средств диалога с пользователем (эта связь, чтобы не загромождать рисунок, не отражена).

Какие практически полезные выводы мы можем сделать из всех этих довольно абстрактных рассуждений о "слоистости" программного обеспечения?

Во-первых, для экономии сил и времени при разработке ПО его различные функции распределены между отдельными слоями. Так работа с печатающим устройством, включая все мелкие детали вплоть до проверки его работоспособности и наличия бумаги, возложена на операционную систему. Причем общая логика работы "прописана" в аппаратно-независимом слое, а управление конкретной моделью печатающего устройства вынесена в отдельный, аппаратно-зависимый, слой. Благодаря такому распределению обязанностей, программист, пишущий прикладную программу, обо всех этих деталях уже не заботится.

Во-вторых, локализация зависимой от оборудования части в отдельном слое системы позволяет легко ее усовершенствовать по мере появления новых видов hardware, практически не затрагивая основного ядра. А прикладное программное обеспечение и вовсе не замечает, например, повсеместную замену матричных принтеров на струйные или лазерные.

В-третьих, подобный способ построения сложных современных программных систем имеет и недостатки: программы, разработанные по многослойной технологии, получаются громоздкими и работают медленнее, чем если бы их писали для конкретной задачи непосредственно в машинных кодах. Впрочем, на это обычно не обращают особого внимания, решая проблему путем постоянного наращивания скорости работы процессоров и объема памяти компьютеров (кстати, перед вами еще один способ незаметно заставить пользователей постоянно обновлять свои компьютеры).

В-четвертых, "отделение" пользователя от компьютера несколькими слоями интеллектуального программного обеспечения позволяет упростить процесс общения человека с компьютером, а значит, расширить круг людей, непосредственно использующих компьютеры в своей повседневной деятельности.

7.4. Главная программа — операционная система

Как уже отмечалось, в рамках нашего курса наибольший интерес представляет системное программное обеспечение, важнейшей частью которого является *операционная система*. Назначение ОС состоит в том, чтобы организовать процесс выполнения задач на ЭВМ, распределяя для этого ресурсы машины (процессорное время, память всех видов, устройства ввода/вывода, программы и данные), а также управляя работой всех ее устройств и взаимодействием с пользователем.

Программная среда ОС позволяет решать несколько очень важных задач, среди которых здесь, прежде всего, хочется выделить возможность настройки компьютера для работы с любым оборудованием, в том числе с появившимся после изготовления машины. В самом деле, если бы алгоритмы обмена информацией между компьютером и внешними устройствами были "жестко" заложены внутри hardware, то набор устройств, с которыми способен взаимодействовать компьютер, был бы фактически предопределен! Наличие операционной системы, напротив, позволяет присоединить к компьютеру любое самое новое внешнее устройство, просто добавив к ОС небольшую специализированную программу — *драйвер*, которая и обеспечит обслуживание данного устройства стандартными средствами операционной системы.

Сформулируем наиболее важные функции операционной системы.

- Организация согласованного выполнения всех процессов в компьютере. Планирование работ, распределение ресурсов и их защита от случайного или преднамеренного повреждения.
- Выполнение обмена с внешними устройствами. Единообразное хранение информации и обеспечение доступа к ней, предоставления справок.

- Запуск и контроль прохождения задач пользователя. Организация взаимодействия задач друг с другом (например, через буфер обмена).
- Реакция на ошибки и аварийные ситуации. Контроль за нормальным функционированием оборудования.
- Обеспечение возможности доступа к стандартным системным средствам (программам, драйверам, информации о конфигурации и т. п.).
- Реализация общения с пользователем.
- Сохранение конфиденциальности информации в многопользовательских системах.
- Организация межмашинного взаимодействия.

Следуя установившейся в нашем курсе традиции рассмотрения всех значительных составляющих вычислительной техники в историческом аспекте, скажем несколько слов в данном ключе и об операционных системах.

Сейчас это может показаться странным, но ОС существовала далеко не всегда: в первых ЭВМ ее вообще не было! Запуском программ, работой машины и всех ее в то время немногочисленных устройств управлял человек. По мере усложнения внешних устройств работать с ними становилось все труднее, что объективно порождало необходимость в какой-то стандартной обслуживающей программе. "Критической точкой" в этом процессе стало появление накопителей на гибких магнитных дисках, доступ к которым, как мы видели в *разд. 5.2.1*, был достаточно трудоемким.

Другой причиной возникновения операционных систем явился переход к многозадачности. Уже в наиболее крупных ЭВМ "на закате" второго поколения принимались меры по ускорению перехода от одной задачи к другой, что существенно повышало эффективность работы машины. Для этого на магнитной ленте из решаемых задач формировался так называемый *пакет*, который обрабатывался как единое целое. Подчеркнем, что задачи пакета все же выполнялись строго по очереди, так что настоящим многозадачным режимом это назвать еще нельзя. Для управления пакетным режимом обработки задач использовалась служебная программа, которую называли *монитором пакетной обработки* [63, 78]. Это был прообраз будущей операционной системы. В ЭВМ третьего поколения, для которых, как мы знаем, был характерен коллективный режим использования, автоматизация процессов прохождения задач через вычислительную машину была еще более усовершенствована, и роль управляющей системы многократно возросла. Появились даже специализированные языки для управления заданиями, с помощью которых оператор сообщал системе необходимые для организации работы сведения. В машинах этого поколения был реализован подлинно многозадачный режим, когда ЭВМ могла одновременно выполнять несколько задач.

Таким образом, рождение операционных систем довольно четко связано с появлением магнитных дисков и ЭВМ коллективного пользования, относящихся к третьему поколению. В зарубежных странах это была серия машин IBM-360, а в нашей стране ее "аналог" — ЕС ЭВМ. Что касается ОС для персональных компьютеров, то здесь первой действительно массовой системой заслуженно считается CP/M, созданная в 1975 году Г. Килдэлом [77].

Примечание

Еще одним доказательством связи ОС с наличием магнитных дисков могут служить комплекты учебной вычислительной техники времен внедрения информатики в школу: КУВТ Yamaha, БК и Корвет. Рабочие места учеников в таких классах могли работать автономно и без операционной системы, используя версию языка Basic в ПЗУ, а также загружаемое с магнитофонных кассет программное обеспечение. Напротив, рабочее место учителя оснащалось дисководом, и на нем обязательно загружалась операционная система (кстати, на "Корвете" тоже использовалась CP/M). Аналогичная картина была и с распространенным бытовым компьютером "Спектрум".

Многие операционные системы (помимо CP/M назовем еще MS-DOS, RT-11, UNIX и Linux [48, 20, 75, 66]) вели диалог с пользователем на экране текстового дисплея. Это был в полном смысле слова диалог, в ходе которого человек и компьютер по очереди обменивались сообщениями: человек вводил очередную команду, а компьютер, проверив ее, либо выполнял, либо отвергал по причине ошибки. Такие системы в литературе принято называть ОС с командной строкой. Типичный пример возможного фрагмента сеанса работы приведен на рис. 7.3.

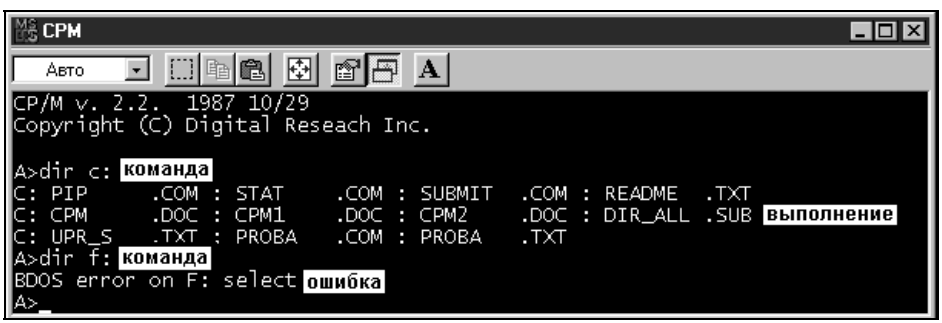


Рис. 7.3. Диалог пользователя и ОС с командной строкой

Мы видим, что пользователь последовательно набрал две команды вывода на экран каталога дисков `dir`, причем первую компьютер выполнил нормально, и на экране появился требуемый список файлов, а вторую "отказался" делать, поскольку оператор ошибочно указал имя несуществующего диска. Очевидно, что подобный способ общения не очень удобен для многих людей, поскольку требует постоянно держать в голове жесткий синтаксис всех необходимых команд и очень внимательно и без опечаток их вводить.

Развитие графических возможностей дисплеев и появление новых устройств ввода привело к коренному изменению принципов взаимодействия человека и компьютера. Командная строка была у рядовых пользователей вытеснена *графическим интерфейсом*, когда объекты манипуляций в ОС изображаются в виде небольших рисунков, а необходимые действия либо выбираются из предлагаемого машиной списка — так называемого *меню*, либо реализуются с помощью определенных манипуляций мышью. При подобном методе диалога для управления компьютером клавиатура почти не требуется. В качестве иллюстрации приведем типичную хорошо понятную картинку, на которой изображено копирование файла с помощью мыши (рис. 7.4).

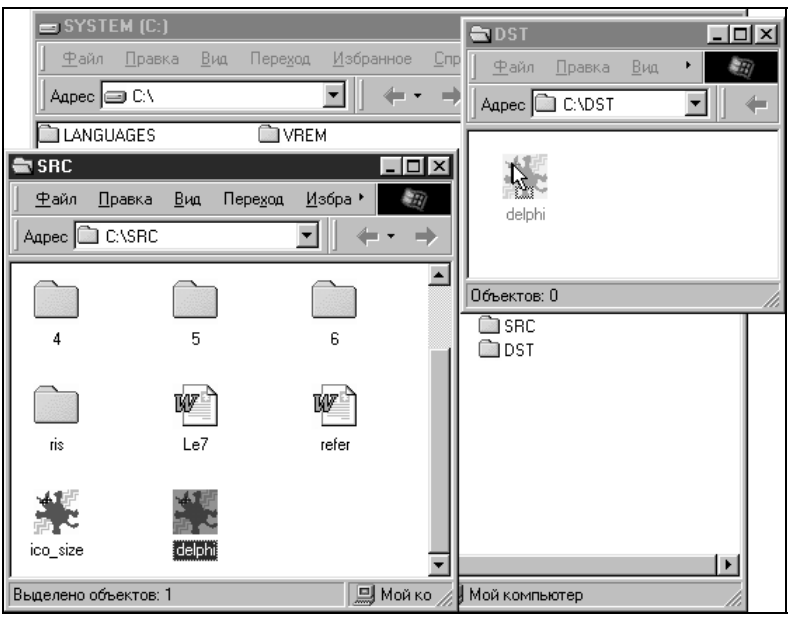


Рис. 7.4. Работа в ОС с графическим интерфейсом

Примерами операционных систем с графическим интерфейсом служат внешне довольно похожие ОС для компьютеров "Macintosh" MacOS и для IBM PC — OS/2 и Windows. Последняя система в нашей стране распространена необычайно широко. Желающим подробнее разобраться в принципах графического интерфейса ОС можно посоветовать обратиться к учебной литературе [34, 26, 81, 76].

С точки зрения изучения основ вычислительной техники интересен еще один важный факт. Наличие собственной логики работы ОС может вносить изменения в принципы обмена информацией с внешними устройствами. Давайте в качестве наглядного примера рассмотрим, как операционная система записывает данные на диски. Мы уже знаем из *разд. 5.2.1*, что минимальной неделимой единицей информации является сектор. Казалось бы,

ОС тоже должна использовать посекторное чтение информации. Но это далеко не всегда так. В наиболее распространенных у нас системах MS-DOS и Windows чтение производится блоками из нескольких секторов, которые называются *кластерами*. Для дискеты кластер состоит всего из одного или двух секторов, а для жесткого диска их входит туда более десятка. Рассмотрим, зачем операционной системе нужны кластеры. Дело в том, что для всех существующих дисковых файлов указанные системы составляют своеобразную "карту" расположения информации (точнее говоря, таблицу, которая называется FAT — File Allocation Table, что в переводе означает "таблица размещения файлов"), которая хранится в начальной служебной области диска. Наличие такой таблицы приводит к существованию двух противоречащих друг другу требований: с одной стороны, чем меньше кластеры, тем их больше на диске и тем больше требуется места для хранения самой таблицы; с другой — чем больше кластеры, тем меньше таблица, но зато тем больше "теряется" дискового пространства, потому что размер файла обязан быть кратен размеру кластера. В частности, если кластер равен 8 Кбайт, то под любой самый маленький файл из нескольких байт отводится именно столько места; файл из 8 Кбайт и 1 байта потребует на диске уже 16 Кбайт и т. д. Таким образом, выбор размера кластера является неочевидным, и он определенно зависит от объема диска.

А теперь несколько слов о том, где нам на практике может пригодиться эта теория. Возьмем, например, типичный файл с пиктограммой Windows размером 766 байт и внимательно посмотрим его свойства (рис. 7.5).

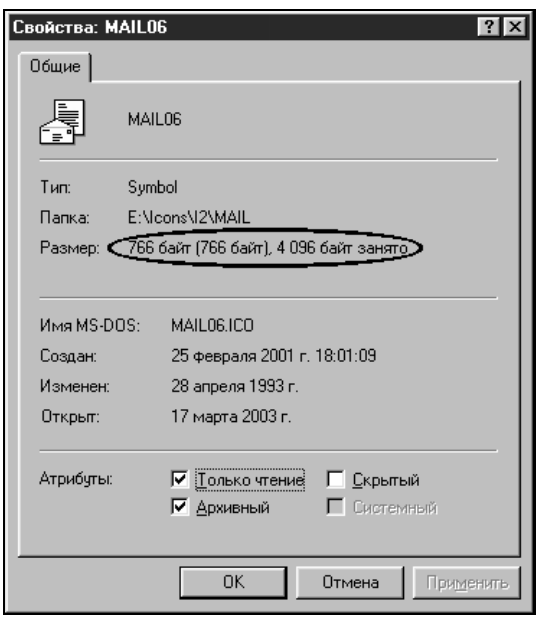


Рис. 7.5. Свойства файла с рисунком значка: отчетливо видна потеря места на диске

Оказывается, на жестком диске данного компьютера он занимает 4096 байт ("коэффициент полезного действия" всего 19 %)! Кроме того, винчестер здесь далеко не самый большой — всего 10 Гбайт, а для бóльших объемов результат будет еще хуже. Немедленно получаем практический вывод: хранение на жестком диске чрезмерно большого количества маленьких файлов может приводить к неоправданно большому расходу места. Хорошим выходом из положения, особенно если информация используется редко, является единый архив таких файлов. Если читателей заинтересовал приведенный выше пример, то впечатляющую дополнительную информацию можно найти на стр. 69—72 книги Лу Гринзоу [13].

Отметим, что кластеры есть не во всех операционных системах. Например, в ОС RT-11 и ей подобных (машины ДВК, БК, УКНЦ) файлы всегда записывались в расположенные последовательно сектора диска, так что положение любого из них определялось его начальным сектором и длиной. При аналогичном подходе FAT-таблица не требовалась, и кластеров, соответственно, не было вообще.

7.5. Порядок загрузки ПО

В момент включения компьютера в его ОЗУ нет осмысленной информации. Поэтому определенный интерес представляет вопрос о том, как попадают туда программы, по которым он работает. Очевидно, что первоначальный загрузчик должен постоянно храниться внутри машины и единственный вид памяти, пригодный для этого, есть ПЗУ.

Мы уже знаем *(см. разд. 3.2.2)*, что при включении компьютера счетчик процессора аппаратно устанавливается на начальный адрес ПЗУ, и загрузка начинается.

7.5.1. Тестирование оборудования

Прежде всего, программа самопроверки, постоянно записанная в ПЗУ (она называется POST — Power-On Self Test, что можно перевести как "самотестирование при включении питания"), ищет подключенное оборудование и убеждается в его работоспособности. Процесс начинается с наиболее важных внутренних устройств компьютера: процессора, минимального объема ОЗУ и некоторых других узлов. До тех пор, пока тест не убедится в подключении исправного дисплея, вся диагностика обнаруженных ошибок осуществляется подачей звуковых сигналов различной длительности. Мне кажется, что если вам никогда не приходилось видеть компьютер, который "пищит и не загружается", то вам очень везло в компьютерной жизни.

После инициализации видеоконтроллера на экране обычно появляется заставка, и последующие диагностические сообщения уже могут выводиться

на экран. Тест продолжается проверкой полного объема ОЗУ (помните, наверное, что при этом отображается на экране?), дисководов, жесткого диска, CD-ROM и всех остальных устройств, подключенных к компьютеру. Всю эту "бурную деятельность" трудно не заметить благодаря характерным резким звукам и поочередному включению индикаторов обращения к соответствующим устройствам. Современные компьютеры в основном используют внешние устройства "Plug-and-Play" *(см. разд. 6.3)*, поэтому в ходе описываемого процесса происходит их программная настройка. Опрос устройств и проверка их работоспособности занимает достаточно длительное время, несмотря на высокое быстродействие компьютера.

Более подробный материал о процессе начального самотестирования читатели смогут найти в уже неоднократно упоминавшейся в нашем курсе книге [15].

7.5.2. Чтение загрузочного сектора

В случае если все оборудование функционирует нормально, происходит переход к следующему этапу — поиску первичного загрузчика операционной системы. Он может находиться на жестком диске, на дискете, на CD-ROM и даже может быть получен с помощью сетевой платы извне. Поэтому компьютер опрашивает перечисленные устройства по очереди, в определенном порядке, до тех пор, пока не обнаружит требуемую информацию (в скобках заметим, что порядок поиска при наличии достаточных навыков может быть легко изменен). Кстати, типичная "мелкая неприятность", которая подстерегает здесь начинающих пользователей, — это наличие в дисководе дискеты без операционной системы. В этом случае, если согласно установленному порядку дисковод опрашивается первым, отсутствие на дискете загрузчика воспринимается как ошибка и выдается соответствующее диагностическое сообщение на английском языке, что порой приводит в замешательство неподготовленных пользователей.

Примечание

Описанная ситуация также представляет потенциальную опасность с точки зрения возможности попадания вируса с дискеты в память машины. Поэтому в литературе убедительно рекомендуется проверять отсутствие дискет в дисководе перед загрузкой или перезагрузкой компьютера. Если все же по невнимательности такое случилось, то после извлечения дискеты лучше перезагрузить компьютер по кнопке сброса, а не продолжать загрузку (подробности см. в [65]).

Напомним читателю, что на рассматриваемом этапе загрузки возможности компьютера еще сильно ограничены, поскольку в ОЗУ по-прежнему нет никаких программ. По этой причине первичный загрузчик всегда находится в самом первом секторе самой первой дорожки (точнее сказать, нулевой, поскольку дорожки, в отличие от секторов, по исторически сложившейся традиции принято нумеровать с нуля) любого диска. Данный загрузочный

сектор был когда-то назван *Boot-сектором*, хотя название это не содержит в себе большого смысла (не хочется повторять здесь рассуждения о "самовытягивании" за шнурки ботинок, положенные в основу терминологии неизвестным юмористом).

Итак, первичный загрузчик, представляющий собой не что иное, как очень маленькую (менее 512 байт для IBM PC) программу дальнейшей загрузки, обнаружен и прочитан в память. Дальнейшие действия машины определяются тем, что прочитано из загрузочного сектора.

7.5.3. Чтение начального загрузчика ОС

Поскольку первичный загрузчик из Boot-сектора очень мал, то он и умеет очень немного — всего лишь найти и прочесть программу последующей загрузки и передать ей управление. Где искать эту программу, уже зависит от загружаемой операционной системы или, точнее говоря, от прочитанного содержимого первичного загрузчика. Так, в RT-11 начальный загрузчик находился в нескольких первых секторах системного диска, а в MS-DOS помещался в специальные системные файлы, которые обязательно должны были в каталоге стоять самыми первыми (вы, конечно, догадываетесь почему — первичный загрузчик слишком мал, чтобы производить в каталоге более сложный поиск!). Так или иначе, после успешного завершения рассматриваемого этапа в ОЗУ компьютера наконец оказывается программный код, способный загрузить операционную систему полностью.

7.5.4. Загрузка операционной системы

В процессе загрузки независимо от конкретной операционной системы в ОЗУ загружается ее резидентное (постоянно находящееся в памяти) ядро, и формируются необходимые условия для его функционирования. И только после этого машина сможет наконец нормально общаться с пользователем. Мы не будем в рамках нашего курса обсуждать подробности данного этапа, поскольку они довольно разнообразны и сильно зависят от конкретной системы. Упомянем для иллюстрации, что формирование ядра системы может происходить динамически в ходе загрузки, как в ОС, разработанных фирмой Microsoft, а может загружаться альтернативным способом, когда ядро предварительно собрано в единый файл, как в ОС семейства UNIX.

Примечания

- Мы рассмотрели процесс, когда на компьютере установлена *единственная* операционная система. На практике, однако, на одном компьютере могут нормально "уживаться" несколько ОС, причем "разной природы", например Windows и Linux. В этом случае дополнительно появляется еще один этап загрузки, в ходе которого пользователь из предложенного меню выбирает нужную ему в данный момент систему. Далее организуется обычная загрузка выбранной ОС.

- Некоторые операционные системы в принципе неспособны стартовать самостоятельно. В таких случаях предварительно загружают какую-нибудь более простую систему, например, MS или DR DOS. Таким образом запускается Novell NetWare или старые версии Windows ниже 3.11 [45].

Несколько слов для тех, кого удивила сложность описанного процесса. Почему загрузка ОС такая многоступенчатая и почему, например, нельзя просто записать начальный загрузчик в то же самое ПЗУ? Технически это не представляет никакого труда, но все дело в том, что тогда мы сможем пользоваться только одной (!) операционной системой, а именно той, загрузчик для которой жестко "зашили" в ПЗУ. Такое нарушение универсальности, разумеется, недопустимо.

7.5.5. Запуск остального ПО

По окончании загрузки операционной системы последняя может автоматически запустить некоторые служебные программы. В RT-11 информация о них извлекается из специального системного файла starts.com, в MS-DOS — autoexec.bat, а в Windows — из системной папки, которая в русифицированной версии называется **Автозагрузка**. В любом случае происходит подготовка компьютера к рабочему состоянию, максимально удобному для конкретного человека. Подчеркнем, что благодаря наличию в современных ОС средств учета и идентификации пользователей начальное состояние компьютера для разных пользователей может быть неодинаковым.

Все дальнейшее программное обеспечение человек, сидящий за компьютером, уже запускает самостоятельно и по своему усмотрению, используя вводимые в той или иной форме (в виде командной строки или мышью) команды операционной системы.

7.6. Какое ПО устанавливать на компьютер

Разумеется, на этот вопрос единого ответа просто не существует. В данном разделе хотелось высказать несколько наиболее общих рекомендаций, не зависящих от интересов и склонностей пользователя конкретного компьютера.

- При подборе ПО, особенно если речь идет об инсталляции в организации, стоит учитывать способ его распространения (вопрос этот уже поднимался в *разд. 7.1*). Если вы не представляете, как вообще можно прожить без дорогих (в прямом смысле этого слова) коммерческих продуктов Microsoft, советую почитать очень информативный лекционный курс [66].
- Широко распространено мнение, что каждая последующая версия ПО обязательно лучше предыдущей. Это далеко не всегда так. Как следует из

разд. 7.3, программное обеспечение строится на базе множества слоев и в каждом из них (а также при взаимодействии двух любых соседних) возможны ошибки и нестыковки. Очевидно, что в недавно вышедшей версии ошибок должно быть больше, чем в версии, находившейся в эксплуатации несколько лет. В то же время, исправление отдельных ошибок, обнаруженных в процессе использования, иногда может приводить к некорректной работе других режимов ПО; в результате при некотором стечении обстоятельств возможно даже увеличение общего количества ошибок по сравнению с первоначальной версией. По-видимому, лучше ставить во главу угла не дату выпуска версии, а ее надежность и качество тестирования. Кстати, а знаете ли вы, что нового дает очередная версия продукта? Многие мои знакомые, которые с гордостью и с хорошо заметным блеском в глазах рассказывают об установке самых последних версий ПО, либо затрудняются с ответом на этот простой вопрос, либо называют нечто яркое, но явно второстепенное. Например, о кошечках или собачках, разгуливающих по просторам текстового документа Word; кстати, потом вдруг оказывается, что их можно при большом желании перенести и в предыдущую версию редактора.

- Если вы используете свой компьютер для серьезной работы (пишете большое количество статей, проводите статистическую обработку результатов многочисленных экспериментов, храните и обрабатываете финансовую информацию, ведете фото- или музыкальный архив), не стоит применять его в качестве полигона для испытаний нового "софта". Поверьте, рано или поздно из-за попытки посмотреть очередную программку смены изображения корзины для мусора в Windows вы потеряете огромное количество действительно ценной и, главное, невосстановимой информации. Иными словами, не стремитесь собрать на своем компьютере все программное обеспечение, какое бывает, а тщательно отбирайте самое необходимое и проверенное.

- Соблюдайте осторожность при установке новых программ из источников, не заслуживающих доверия (к их числу относятся и многие сайты Интернета). Помимо возможности незаметно "заполучить" компьютерный вирус, есть еще риск, что предлагаемое ПО не совсем соответствует вашему компьютеру или плохо взаимодействует, например, с установленным у вас резидентным антивирусным монитором. Лучше все-таки использовать хорошо проверенное или рекомендуемое авторитетными журналами программное обеспечение.

- Старайтесь по возможности не устанавливать ПО, которое требует нестандартных режимов работы или настроек машины (если, конечно, вы не принадлежите к тем редким людям, которым процесс восстановления работоспособности компьютера доставляет удовольствие). Особенно часто к подобного рода эффектам приводит установка игровых программ,

авторов которых порой заботит только нормальное функционирование собственной программы.

7.7. Еще раз о роли программирования

В последнее время в связи с процессами, о которых говорилось в *лекции 1*, многократно усилились попытки полностью "изгнать" знакомство с системами программирования из содержания курса информатики. Поэтому, завершая обсуждение различных категорий ПО, давайте посмотрим, насколько полезны подобные навыки для тех, кто работает на компьютере.

Начнем с того, что знание основ программирования позволяет понимать логику работы компьютера, чего, по мнению автора, уже должно быть достаточно. Кроме того, программирование развивает строгость и логику мышления, которой нам так сейчас не хватает в окружающей жизни (все эти противоречащие друг другу, подчас абсолютно лишенные всякого смысла законы и инструкции, попытки по-разному подсчитать количество избраний на политическую должность и многое-многое другое, с чем мы каждодневно сталкиваемся).

Зачем пытаться программировать самому, если есть столько готовых программ? Подобный подход А. Г. Кушниренко в одном из интервью метко назвал азиатским [53]:

> "Время вечно и не очень ценится, информации под руками нет, до ближайшего источника информации три дневных перехода (переезда), теорему легче доказать, чем раздобыть книжку с ее доказательством. Так поступают индийские, русские, китайские специалисты".

Да, американцы делают по-другому, и вы знаете как — вызывают (легко доступную у них!) квалифицированную помощь при любом затруднении. И если на производстве "нельзя сказать, что один подход лучше, чем другой" [53], то в обучении определенно можно! Сильным и ничем не заменимым достоинством "азиатского" подхода с образовательной точки зрения является то, что "проблема рассматривается содержательно" [53], а значит, ученик развивает свои мыслительные способности. Если школьника этому своевременно не учить, то он вряд ли вообще сможет самостоятельно решать какие-либо проблемы, кроме самых тривиальных (вспомните пословицу по поводу лежачего камня). И будет поступать как пользователь, про которого мне недавно рассказывали: тот не мог записать файл на дискету, поскольку на компьютере сломался дисковод. Но ему и в голову не пришло, что можно воспользоваться дисководом другого компьютера, который стоял на соседнем столе и был соединен с первым локальной сетью! Неужели мы хотим прийти к этому повсеместно?

Наконец, программирование и само пытается постоять за себя, "вылезая" там, где его, казалось, не должно быть вовсе. Например, возьмем текстовый

редактор Microsoft Word. Вроде бы при работе с ним программирование уж точно не нужно. Но нет — квалифицированные люди почему-то пишут специальные команды для обработки текста — "макросы" (и, между прочим, на многократно обруганном языке Basic, хоть и с модной приставкой Visual). Или система Maple, призванная избавить пользователей от необходимости проводить трудоемкие математические преобразования — там тоже можно получить результат безо всякого программирования. Однако при попытке решить задачу посложнее приходится многократно повторять одни и те же действия и, устав нажимать клавишу <Enter>, грамотный пользователь снова обращается к циклу.

Сказанное не следует понимать как попытку обосновать невозможность работы на компьютере без программирования и призыв к приоритетному его изучению. Но не стоит впадать и в другую крайность, полностью отрицая, что знание основ программирования позволяет более квалифицированно использовать компьютер и получать лучшие результаты.

И еще. Каждый, кто умеет программировать, обязательно пользуется текстовым редактором и копирует файлы, но не каждый, кто пользуется текстовым редактором и копирует файлы, умеет программировать.

7.8. Выводы

Итак, программное обеспечение играет в работе ВТ не меньшую роль, чем собственно компьютерное оборудование. Приведем выводы, которые можно сформулировать по этому поводу из данной лекции.

- В настоящее время использование компьютера немыслимо без многочисленного программного обеспечения. ПО позволяет приспособить стандартное компьютерное оборудование к запросам самых разных по интересам и потребностям пользователей. Наличие мощного ПО существенно облегчает общение человека с компьютером и многократно ускоряет решение стоящих перед пользователем задач.
- Современное программное обеспечение разделено на целый ряд слоев, каждый из которых выполняет вполне определенные функции.
- За рост сложности ПО приходится платить снижением надежности его работы, а также постоянной потребностью в повышении производительности процессора и объема всех видов памяти.
- Аппаратно-зависимую часть программного обеспечения стремятся локализовать отдельно и сделать эту часть минимальной, что облегчает подключение нового оборудования.
- Главной служебной программой является операционная система. Без нее нельзя запустить на компьютере практически ни одну прикладную программу. Кроме этого, ОС имеет большое количество других функций,

включая диалог с пользователем. По мере развития периферийных устройств диалог с компьютером становится все более удобным для человека.

- Процесс загрузки программного обеспечения при включении компьютера является многоступенчатым, что обеспечивает значительную гибкость и возможность усовершенствований в будущем.
- В связи с огромным разнообразием ПО его выбор является нетривиальной проблемой. При ее решении приходится учитывать множество различных факторов.

7.9. Вопросы для осмысления

1. Согласны ли вы с теми утверждениями о роли программного обеспечения, которые высказаны в лекции? Приходилось ли вам наблюдать случаи, когда не удавалось воспользоваться работающим компьютером из-за отсутствия необходимых программ?
2. Какие виды программного обеспечения вы знаете? Каковы их функции?
3. К какому виду ПО вы отнесете следующие программы: архиватор WinZip, Norton Commander, текстовой редактор Блокнот, компьютерные игры?
4. Прочитайте в помощи к любому коммерческому программному продукту (например, подобный раздел имеется в каждой программе фирмы Microsoft) материалы, связанные с лицензионным соглашением. Что является признаками законной установки на компьютере ПО, распространяемого коммерческим путем?
5. Какие формы распространения программного обеспечения вам известны?
6. Помните ли вы, какое программное обеспечение установлено на вашем компьютере? Насколько тщательно вы его подбирали?
7. Подробно изучите рисунок в *разд. 7.3*. Как вы думаете, все ли слои ПО там отражены? (Указание: в случае затруднения *см. разд. 7.10.5*.)
8. Каковы достоинства и недостатки разделения ПО на отдельные слои?
9. Приходилось ли вам когда-нибудь добавлять к компьютеру новые устройства? Вспомните, какая программная настройка при этом потребовалась? Где вы брали драйверы для новых устройств (были в комплекте, скопировали у знакомых, "закачали" из Интернета)?
10. С какими операционными системами вы работали и какие видели, как они работают?
11. Назовите и объясните как можно большее количество функций операционной системы. Знаете ли вы какие-либо функции, не упомянутые в тексте лекции?

12. Насколько закономерно существование операционных систем?
13. Как вы думаете, есть ли операционная система на компьютере, который весь день считывает штрихкоды и подсчитывает стоимость товара в крупном магазине?
14. Приходилось ли вам пользоваться командной строкой при работе с операционной системой? Можно ли ввести командную строку в Windows?
15. Внимательно пронаблюдайте порядок загрузки ПО вашего компьютера. Постарайтесь заметить максимальное число деталей и идентифицировать их с описанными в *разд. 7.5* этапами загрузки. Какие программы автоматически запускаются в вашем компьютере?
16. Приходилось ли вам устанавливать более новые версии ПО на компьютер? Что из перечисленного вы при этом наблюдали: конфликт с предыдущим ПО, неработоспособность установленной версии по тем или иным причинам, требование ввести серийный номер продукта, существенное уменьшение свободного места на винчестере, изменение скорости работы программы, появление новых удобных возможностей при работе с ПО, изменение расположения органов управления программы и пунктов меню и т. п.?
17. Каково ваше личное отношение к программированию?

7.10. Любопытные эксперименты

7.10.1. Какие программы установлены на компьютере?

Большинству пользователей нравится делать что-нибудь на компьютере, многие даже по-своему любят его. Отсюда чаще всего следует тщательный подбор программного обеспечения: те, кто действительно проводят время за компьютером с удовольствием, обычно знают на память не только все установленные у них программы, но даже соответствующие номера версий. Тем не менее время от времени вопрос, вынесенный в заголовок, все же возникает, скажем, когда вы садитесь за "чужой" компьютер. Итак, как можно узнать, какие программы установлены на компьютере?

Цель эксперимента. Научиться определять, какие программы установлены на компьютере.

Самый очевидный метод состоит в том, чтобы нажать "главную кнопку" Windows на экране дисплея — **Пуск** и выбрать в появившемся меню пункт **Программы**. Вследствие простоты этого действия, которое вы, скорее всего, проделывали уже неоднократно, мы не будем описывать подробности.

Зададимся вопросом, все ли программы мы увидим таким путем? Абсолютно очевидно, что нет, хотя бы потому, что состав меню можно редактиро-

вать, а значит, изменять по своему усмотрению. Но наиболее интересна не возможность произвольного изменения списка программ; в конце концов, эти действия мы выполняем сами, а стало быть, обо всех подобных изменениях знаем. Гораздо менее тривиален ответ на вопрос: все ли устанавливаемые на компьютер программы автоматически попадают в главное меню. Вспомнив, как могут устанавливаться программы, мы без особых колебаний ответим и на этот вопрос отрицательно. В самом деле, по "старым добрым" традициям MS-DOS некоторые программы переносятся простым копированием совершенно так же, как копируются файлы с текстовой или звуковой информацией. Поскольку Windows не в состоянии "самостоятельно" надежно определить назначение копируемых файлов, он, разумеется, не анализирует подобные операции на предмет занесения программ в главное меню. В подобной ситуации добавление в меню пользователю придется проделать самому.

Зато сложные программные пакеты, существенным образом использующие возможности среды Windows, обязательно устанавливаются при помощи специальной процедуры инсталляции. В ходе обсуждаемого процесса решается целый ряд задач: создание необходимой иерархической структуры каталогов, размещение в ней требуемым образом всех файлов программы (обычно происходит их распаковка из архива), нужная коррекция путей для конкретного жесткого диска, добавление необходимой информации о программе в системную базу данных — *реестр*, и многое другое. Для нас сейчас наиболее интересно, что инсталлятор автоматически создает ссылки на программы пакета в главном меню (в последнее время некоторые из них стали разрешать пользователю пропустить данное действие).

Итак, при наличии специально подготовленного процесса инсталляции обязательно предусмотрена возможность отображения устанавливаемой программы в главном меню кнопки **Пуск**.

Интересно заглянуть еще в одно место, куда заносится информация обо всех корректно инсталлируемых программах. Выберите в меню кнопки **Пуск** пункты **Настройка** и **Панель управления**, а затем откройте служебную папку **Установка и удаление программ**. Вы увидите список всех пакетов, которые относятся к этой категории (рис. 7.6).

Примечание

Прежде чем удалять каталог со ставшей ненужной по тем или иным причинам программой, обязательно загляните в обсуждаемый список. Если программа в него входит, то ее обязательно надо деинсталлировать именно отсюда! Если же интересующее название в списке не обнаружено, дополнительно поищите для него программу с названием Uninstall. И только при отсутствии и того, и другого удаляйте папку обычным образом.

В последнее время все большее количество программ устанавливается с помощью полноценного инсталлятора. Его наличие во всех крупных интернет-библиотеках ПО обязательно отмечается в виде отдельной характеристики.

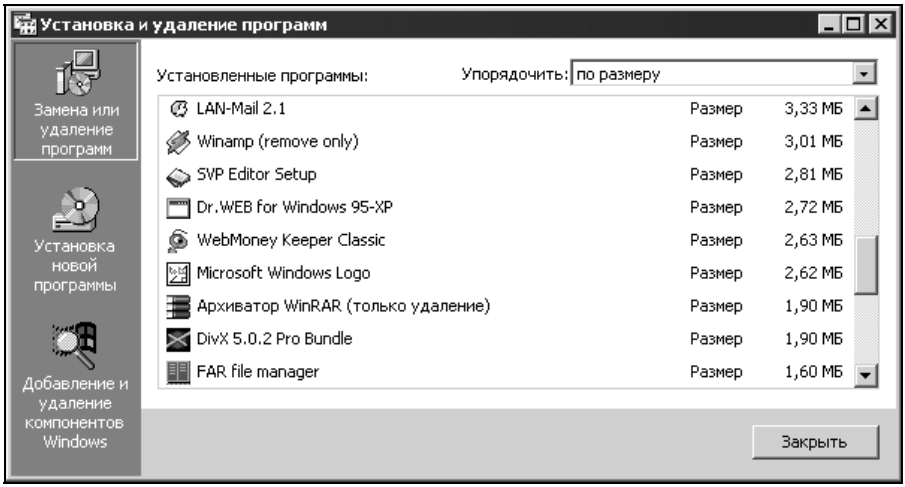


Рис. 7.6. Отображение имеющегося программного обеспечения в окне установки и удаления программ

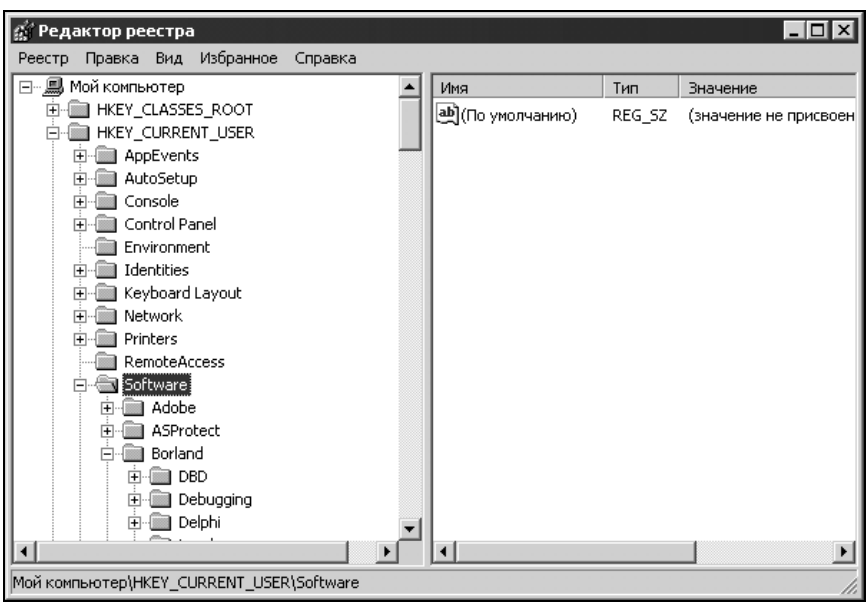


Рис. 7.7. Информация об установленном программном обеспечении в реестре

Наконец, заглянем в еще одно, наиболее труднодоступное место, где на вашем компьютере регистрируется программное обеспечение Windows. Речь идет о реестре — базе данных обо всем, что известно Windows. Наличие единой базы, где можно найти сведения о виде пиктограммы "Мой компьютер", вместе с тонкими настройками hardware часто подвергается критике,

как элемент потенциальной ненадежности. Но так или иначе реестр продолжает существовать. Для его просмотра и редактирования имеется специальная утилита RegEdit. Поскольку в компании Microsoft считают, что рядовой пользователь не должен работать с реестром, на эту программу нет никаких явных ссылок. Вызвать ее проще всего через директиву **Выполнить** кнопки **Пуск**: выберите ее и напечатайте название программы RegEdit (помните, как мы это делали в эксперименте, описанном в *разд. 2.5.4*?) Затем в появившемся окне откройте папку HKEY_CURRENT_USER и найдите папку SOFTWARE (рис. 7.7).

В ней-то и находятся сведения обо всех зарегистрированных "штатным образом" программах для конфигурации данного пользователя. Аналогичные данные, но уже для всех пользователей, имеются также по пути HKEY_USERS\.DEFAULT\SOFTWARE.

Вывод. Таким образом, в случае полноценной инсталляции каждая устанавливая на вашем компьютере программа фиксируется операционной системой.

7.10.2. Определение размера кластера

В *разд. 7.5* лекции сказано, что операционные системы Windows и MS-DOS при хранении информации на диске используют кластеры.

Цель эксперимента. Установить размер кластера на жестком диске вашего компьютера или на дискете.

Это очень простой эксперимент, фактически те, кто внимательно прочитал описание рис. 7.5 в тексте лекции, уже знают, что надо делать. Выберем короткий файл, размер которого заведомо меньше одного сектора, т. е. 512 байт. Например, можно просто обратиться к прилагаемой к книге дискете и в папке 7, относящейся к данной лекции, взять файл EICAR.com, предназначенный для одного из следующих экспериментов (его длина 68 байт). Посмотрим свойства выбранного файла с помощью правой кнопки мыши. Размер кластера будет указан рядом со словами "занято на диске" или аналогичными.

На всякий случай напомним, что размер кластера на дискете отличается от размера кластера на жестком диске, поэтому для одного и того же файла результаты зависят от местоположения. Для дискеты вы должны получить 512 байт, что соответствует одному сектору (раньше использовались дискеты, для которых размер кластера был вдвое больше). А вот для жесткого диска ответ предсказать не берусь, уверен только, что он как минимум в несколько раз превышает таковой для дискеты.

Вывод. Кластеры действительно существуют, и теперь вы знаете их размер для своего диска.

7.10.3. Несколько экспериментов с именами файлов

Как все хорошо знают, обращение к файлам с информацией во всех современных операционных системах происходит по именам. Полюбопытствуем, как система работает с именами файлов.

Цели эксперимента. Разобраться, как система ищет файлы; посмотреть, с чем связаны ограничения в именах файлов.

Начнем с несложной программы, которая выводит на экран перечень всех каталогов, находящихся на вашем диске C: (листинг 7.1).

Листинг 7.1. Программа, выводящая на экран перечень всех каталогов, находящихся на диске C:

```
PROGRAM dir;
USES  WinDos;
CONST Attrs=faDirectory;   {каталоги}
VAR   DirInfo: TSearchRec; {данные о файле}
BEGIN WRITELN; WRITELN;
      FindFirst('c:\*.*', Attrs, DirInfo);
      WHILE DosError = 0 DO
            BEGIN IF DirInfo.attr=Attrs
                     THEN Write(DirInfo.Name:16);
                  FindNext(DirInfo);
            END
END.
```

Константа `Attr` задает атрибуты интересующих нас объектов (в нашем случае программа будет работать со значением `faDirectory`, которое соответствует каталогам), а переменная `DirInfo` резервирует память под информацию о найденных объектах. Традиционно поиск необходимых объектов в каталоге осуществляется двумя функциями: `FindFirst` и `FindNext`. Первая из них находит так называемое "первое вхождение" заданного шаблона, проще говоря, фиксирует первый объект заданного типа, имя которого удовлетворяет передаваемому в качестве параметра шаблону. Следующая функция, которая обязательно вызывается после `FindFirst`, ищет следующий объект. Делается это неоднократно, до тех пор, пока вызов `FindNext` происходит результативно, т. е. признак ошибки `DosError = 0`. Таким образом, с помощью этих двух функций удается распечатать весь каталог.

Для каждого найденного файла в структуру `DirInfo` формируется запись, содержащая полную информацию о файле. В нее входят имя файла, его размер, время создания и атрибуты. Все эти сведения копируются из 32-байтовой записи в каталоге диска. Для MS-DOS она имеет структуру, описанную в табл. 7.2.

Таблица 7.2. *Структура записи каталога в MS-DOS по данным [61]*

Поле	Смещение	Описание	Размер (байт)	Формат
1	0	Имя файла	8	Символы ASCII
2	8	Тип файла	3	Символы ASCII
3	11	Атрибуты	1	Биты (код)
4	12	Служебное поле	10	Не используется
5	22	Время создания	2	Слово (код)
6	24	Дата создания	2	Слово (код)
7	26	Начальный кластер	2	Слово
8	28	Размер файла (байт)	4	Целое

Из табл. 7.2 видно, что для каждого файла кроме его имени и расширения в 5-ти полях записи определенным образом хранятся наиболее важные характеристики файла (одно поле не используется). Запомним для следующего эксперимента, что в поле 7 указывается ссылка на первый кластер файла, благодаря чему система может начать считывание файла.

Фиксированный размер первых двух полей определяет известный стандарт MS-DOS, который часто сокращенно называют 8.3. Сейчас он уже устарел, а во всех версиях системы Windows разрешено употребление так называемых длинных имен: до 255 символов включая знаки точки и пробелы. Так что текст **Мой отчет за 2003 год** вполне может быть именем текстового документа или соответствующей папки.

По поводу длинных имен предлагаю попутно проделать следующий довольно забавный эксперимент (рис. 7.8). Создайте на диске C: папку и назовите ее **Папка для теста**. Поместите в нее любой растровый рисунок, который переименуйте в тест.bmp. Выберите в меню кнопки **Пуск** пункт **Выполнить** и напишите вызов графического редактора mspaint с целью открыть наш рисунок с помощью следующей строки:

mspaint c:\Папка для теста\тест.bmp

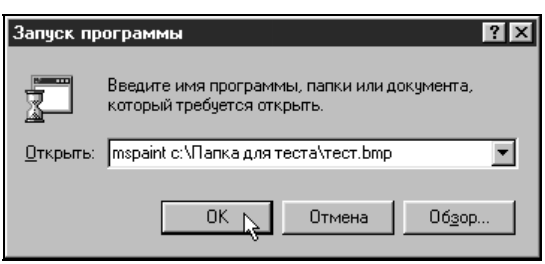


Рис. 7.8. Эксперимент с длинным именем, содержащим пробелы

Примечание

Набранная только что строка имеет традиционную для всех ОС структуру

<команда> <параметры команды>

причем для разделения указанных частей используется пробел. Кстати, когда вы инициируете открытие документа с помощью мыши, система внутри себя генерирует подобную строку самостоятельно.

Результат выполнения описанного выше простого, казалось бы, действия будет довольно неожиданным (для систем Windows 98 и Windows 2000 он представлен на рис. 7.9).

Рис. 7.9. Неожиданный результат эксперимента с длинным именем

И хотя причина странности, в общем, понятна (пробел в имени файла воспринимается в качестве разделителя якобы в списке нескольких параметров командной строки), сообщение на рис. 7.9 трудно объяснить логически. Для восстановления "разумного" поведения системы достаточно имя рисунка заключить в кавычки:

mspaint "c:\Папка для теста\тест.bmp"

Тем читателям, которых заинтересовали проблемы, связанные с переходом на длинные имена, стоит непременно почитать "Не слишком краткую лекцию о длинных именах файлов" в книге Л. Гринзоу [13].

Выводы. Каталог диска содержит специальные записи, каждая из которых хранит информацию об одном файле. В них отражается имя файла, его наиболее важные характеристики, а также информация, необходимая для нахождения файла на диске (подробнее см. следующий эксперимент). Формат записей существенным образом зависит от операционной системы.

7.10.4.* Изучение расположения файлов на дискете

Мы уже упоминали при обсуждении предыдущего эксперимента, что каждая запись в каталоге содержит поле, указывающее на начало дискового объекта. Если система всегда хранит информацию в последовательных секторах (подобную организацию, как уже отмечалось ранее, имела RT-11), то знания

начала файла и его длины вполне достаточно для успешного считывания. А как быть с MS-DOS, где, напротив, заложена возможность расположения информации в несмежных секторах диска? Этому и посвящается следующий эксперимент.

Цель эксперимента. Разобраться, как система "собирает" все сектора, принадлежащие данному файлу.

Для проведения экспериментов потребуется отформатированная дискета. Мы будем записывать на нее не очень длинные файлы (2—3 Кбайт) и следить за их размещением. Разумеется, принципы записи информации на пустую и на заполненную дискету принципиально не различаются, но при сложной структуре файлов на дискете разобраться будет труднее.

Немного теории. Как обычно, перед сложным экспериментом неплохо хотя бы кратко познакомиться с теорией. Мы уже знаем из предыдущего эксперимента, что о каждом файле в каталог диска делается специальная запись (см. табл. 7.2). В ней отражается имя файла и его характеристики, в том числе и физическое местоположение файла на диске. В частности, для системы MS-DOS в каталоге указан самый первый кластер файла.

Примечание

Конечно, такой способ не является единственно возможным. Например, в операционной системе CP/M, которая непосредственно предшествовала MS-DOS, в запись каталога заносилась информация *обо всех* секторах файла (а не только о первом). Вы спросите, как удавалось разместить информацию о файлах разной длины в фиксированную по размеру запись каталога? Ответ прост: запись была рассчитана на файл до 16 Кбайт включительно, а если он оказывался больше, то в каталоге заводилось несколько записей.

Расположение остальных секторов операционная система находит с помощью специальной таблицы, которая называется *FAT* — File Allocation Table (таблица размещения файлов). В этой таблице каждому кластеру выделено специальное место (образно говоря, отведена ячейка) для информации о состоянии данного кластера. Свободный нераспределенный в файл кластер в FAT-таблице отображается нулем, сбойный и потому непригодный для хранения информации специальным кодом FF7. Когда тот или иной кластер включается в состав файла, в соответствующую "ячейку" таблицы заносится номер следующего кластера файла. Если же кластер оказался последним, то ссылка уже не требуется и такой кластер помечается особым служебным кодом FFF (кластера с таким большим номером, как, впрочем, и с номером FF7, не существует).

Используя описанные принципы, система может однозначно описывать расположение частей файла на диске.

Примечание

При нормальном функционировании системы все это "хозяйство" работает слаженно и безупречно. Тем не менее, в случае сбоев или порой при некорректных действиях пользователя структура FAT-таблицы может нарушаться. Типичным примером подобного рода является некорректное выключение компьютера (при пропадании питания или при неквалифицированных действиях пользователя), когда на диске часто остаются файлы, работа с которыми не завершена. В такой ситуации некоторые кластеры в FAT-таблице могут быть помечены как занятые, но принадлежать незавершенным, а значит, неработоспособным файлам. Эту и многие другие проблемы такого рода способна решить утилита проверки дисков; довольно часто при некорректном завершении работы ОС Windows в ходе следующей загрузки она запускается автоматически.

Для обобщения полученных знаний советую внимательно рассмотреть рис. 7.10, показывающий логическое устройство дискеты.

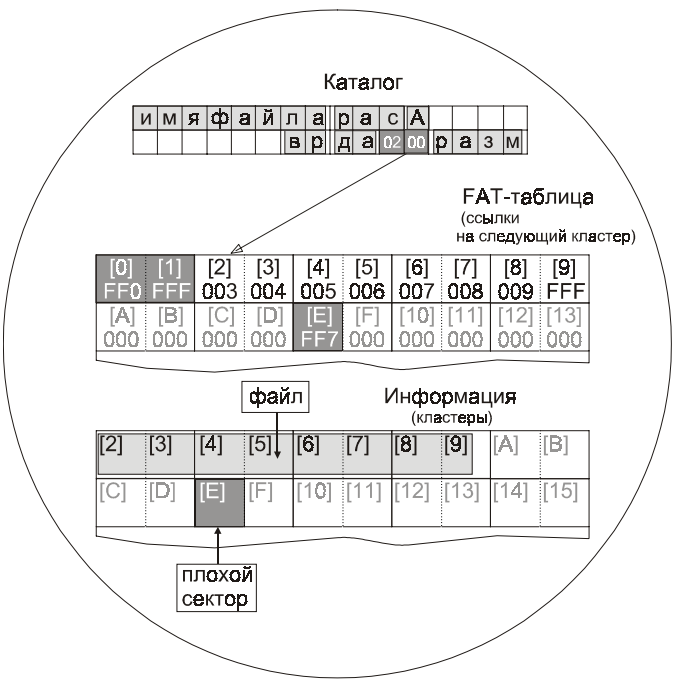


Рис. 7.10. Логическое устройство дискеты MS-DOS

На рисунке изображена одна запись каталога, на которой помечены все основные поля: имя файла, его расширения (подпись **рас**), атрибуты файла (**А**), время (**вр**) и дата (**да**) создания файла, а также его размер (**разм**). Наиболее интересное для нас сейчас поле с первым кластером файла содержит два байта 02 и 00, что соответствует числу 0002. Перестановка старшего и

младшего байта является еще одним следствием архитектуры "little-endian", о которой мы подробно говорили ранее в *разд. 4.2.2.*

Запись каталога ссылается на FAT-таблицу, часть которой также изображена на рис. 7.10. В квадратных скобках указаны номера кластеров, а во второй строке ячеек таблицы приводится их содержимое, т. е. номер следующего кластера файла. Поскольку в MS-DOS на дискетах 1,44 Мбайт применяется 12-битовый FAT, то каждый элемент таблицы занимает 1,5 байта и элементы "естественным" образом группируются в "пары" — по 3 байта в каждой.

Примечание

Первая пара элементов таблицы является служебной: там записан тип формата диска (в нашем случае он равен F0). Именно поэтому нумерация кластеров в таблице начинается с двойки.

"Пары" кластеров пакуются в таблицу довольно необычным образом. Например, имеющаяся на рис. 7.10 пара 003 и 004 в виде последовательности 3 байт выглядит так: *03 40* 00 (курсивом выделены цифры, относящиеся к первому элементу). Подобный нетривиальный способ запаковки реализован ради эффективности обработки данных: они хранятся, как удобно машине, а не как наглядно с точки зрения человека. При необходимости программное обеспечение всегда сможет распаковать данные и преобразовать их к виду, удобному пользователю компьютера для обозрения (как на рис. 7.10).

Перейдем теперь непосредственно к проведению эксперимента. Для контроля за содержимым дискеты потребуется читать ее различные сектора. Мы уже умеем решать эту задачу *(см. разд. 5.6.1)*, поэтому возьмем за основу листинг 5.1 и дополним его необходимыми действиями. Получим новую, более длинную программу, которая приведена в листинге 7.2.

Листинг 7.2. Программа, иллюстрирующая, как система находит файл

```
PROGRAM disketa;
VAR buf:ARRAY [0..511] OF CHAR;
    ns,nc,s0,cat,inf,i,k,m:INTEGER;

{Находящиеся здесь процедуры printChar, hexPrint и printHalf
см. в листинге 5.1}

PROCEDURE ReadSector(n:integer); {читает сектор диска по n}
BEGIN ASM lea bx,buf {адрес буфера}
          mov dx,n   {номер сектора}
          mov cx,1   {количество секторов}
          mov al,0   {диск A:}
          int 25h
          pop cx
      END;
END;
```

```
FUNCTION NextCluster(p:INTEGER):INTEGER;
{по FAT-таблице определяет кластер, следующий за данным;
см. книгу: Р.Джордейн. "Справочник программиста
персональных компьютеров типа IBM PC и XT", с.289}
VAR w:INTEGER;
BEGIN IF p>255 THEN
        BEGIN WRITELN('Некорректный номер кластера');
              NextCluster:=$FFF; EXIT
        END;
      ASM lea bx,buf   {адрес буфера}
          mov ax,p     {предыдущий кластер}
          mov cx,ax    {копия1}
          mov dx,ax    {копия2}
          shr dx,1     {сдвиг, т. е. копия2 / 2}
          add cx,dx    {копия1 + копия2 / 2 = 1.5*p}
          add bx,cx    {добавляем, как смещение}
          mov dx,[bx]  {получаем 2 байта с этого места}
          test ax,1    {номер кластера нечетный?}
          jnz @1       {уходим, если да}
          and dx,0FFFh {сбросим старшую цифру}
          jmp @2       {обход}
      @1: mov cl,4     {готовим сдвиг на 4 вправо}
          shr dx,cl    {удаляем последнюю цифру}
      @2: mov w,dx
      END;    NextCluster:=w;
END;

BEGIN ReadSector(0); {##### Характеристики диска #####}
      WRITELN('Размер сектора — ',
               ORD(buf[12])*256+ORD(buf[11]),' байт');
      WRITELN('Размер кластера — ',ORD(buf[13]),' сект.');
      WRITELN('Секторов на дорожке — ',ORD(buf[24]));
      WRITELN('Всего секторов — ',ORD(buf[20])*256+ORD(buf[19]));
      WRITE('Код формата диска — '); hexPrint(ORD(buf[21]));
      WRITELN;WRITELN;
      cat:=2*ORD(buf[22])+1; {начало каталога: 2 FAT + 1}
      inf:=cat+ORD(buf[17])*32 div 512; {начало информации: +каталог}
      ReadSector(cat); {##### Каталог #####}
      WRITELN('сектор ',cat,' (начало каталога)'); printHalf(0);
      WRITE('Введите номер строки '); READLN(ns);
      k:=(ns-1)*16+26; {позиция первого кластера файла}
      nc:=ord(buf[k])+ord(buf[k+1])*256; {номер кластера}
      WRITE('Файл — '); {выведем имя}
      FOR i:=0 TO  7 DO WRITE(buf[k-26+i]); WRITE('.');
      FOR i:=8 TO 10 DO WRITE(buf[k-26+i]);WRITELN;
```

```
      WRITELN('Первый кластер файла – ',nc);
      ReadSector(1); {##### FAT #####}
      WRITELN('сектор 1 (начало FAT)'); printHalf(0);
      s0:=inf-2+nc; {начальный сектор файла}
      WRITE('Цепочка FAT: ');
      REPEAT hexPrint(nc); nc:=NextCluster(nc)
      UNTIL  nc=$FFF; READLN;
      WRITELN('Первый сектор файла – ',s0);
      ReadSector(s0); {##### Файл #####}
      printHalf(0); READLN
END.
```

Обратите внимание, что несколько процедур, взятых из листинга 5.1 без изменения, из текста программы удалены.

Рассмотрим подробнее, как устроена программа для эксперимента. Несмотря на то, что она кажется огромной и сложной, на самом деле она просто состоит из нескольких похожих и относительно независимых частей. Каждая из них считывает один сектор (это происходит в процедуре `ReadSector`, устройство которой разбиралось ранее в *разд. 5.6.1*), обрабатывает его и выводит результаты на экран.

С точки зрения обсуждаемой темы наиболее интересной является устройство процедуры `NextCluster`, которая каждый раз обеспечивает нахождение следующего кластера файла. Остановимся на ней подробнее.

Данная процедура является тем редким случаем, когда на Ассемблере программа короче и проще, чем на языке высокого уровня. Парадокс объясняется тем, что при разработке устройства FAT-таблицы простота и, следовательно, эффективность обработки были взяты за основу, а наглядность и удобство объяснения были принесены в жертву.

Мы уже знаем, что для дискет таблица FAT является 12-битовой, так что под каждый кластер отводится 1,5 байта (лучше сказать, информация о 2 кластерах занимает объем 3 байта). Поэтому в первую очередь определяются координаты той пары байт, в которой записан интересующий нас кластер. Для этого номер исходного кластера `p` умножается на 1,5 достаточно экстравагантным, но эффективным способом. В вычислениях используется очевидное тождество 1,5 $p = p + 0{,}5\,p$, причем второе слагаемое получается путем сдвига `p` вправо на один разряд. Возможно, наиболее внимательные читатели сумели заметить некоторое сходство используемого алгоритма с умножением на 10, разбиравшимся в *разд. 3.3.4*. Получившийся в итоге результат прибавляется к началу FAT-таблицы и оттуда в регистр `DX` извлекается, наконец, необходимая информация о нужном кластере. Но это, увы, еще не конец, поскольку из прочитанных 16 бит нам требуется только 12, а значит, ненужные 4 бита требуется сбросить. Поскольку кластеры "пакуются" парами в 3 байта, то очевидно, что для четного и для нечетного номера потре-

буются разные действия. Четность анализируется с помощью команды `test ax,1`, которая выделяет младший бит номера кластера. Если последний был нечетным, то младший бит равен 1 и происходит переход к удалению последних 4 бит с помощью сдвига вправо. В противном случае первые 4 бита сбрасываются с помощью логической операции И.

Рассмотренная процедура `NextCluster` является очень характерным примером программирования в машинных командах, поэтому с этой точки зрения она очень интересна. Если же для читателей эта сторона не очень важна, то можно просто считать, что данная процедура по номеру кластера извлекает из FAT-таблицы номер следующего, а как именно это происходит — оставить "ученым и инженерам".

Остается разобрать основную программу. Как обычно у нас бывает, она уже не очень сложная, т. к. все детали вынесены в процедуры. Прежде всего, считывается нулевой (загрузочный) сектор, в котором нас сейчас интересует находящаяся там таблица с характеристиками дискеты. В таблице хранится следующая важная информация (табл. 7.3).

Таблица 7.3. *Таблица характеристик диска в Boot-секторе MS-DOS по данным [61]*

Смещение	Длина	Описание
0	3 байта	Переход к загрузчику
3	8 байт	Идентификатор системы
11	1 слово	Размер сектора (обычно 512 байт)
13	1 байт	Размер кластера (обычно 1 или 2 сектора)
14	1 слово	Количество резервированных секторов (для дискеты 1)
16	1 байт	Количество копий FAT (для дискеты 2)
17	1 слово	Количество файлов в корневом каталоге
19	1 слово	Общее количество секторов на диске (2880 для 1,44 Мбайт)
21	1 байт	Условный код формата
22	1 слово	Размер FAT
24	1 слово	Количество секторов на дорожке
26	1 слово	Число сторон диска (для дискеты 2)
28	1 слово	Количество специальных секторов

Наша программа извлекает наиболее интересные характеристики из прочитанной с дискеты таблицы и выводит их на экран. Попутно заметим, что

для нас сейчас наиболее информативен байт со смещением 13, показывающий количество секторов в кластере — эксперимент покажет, что для стандартной дискеты 1,44 Мбайт кластер равен сектору.

С использованием некоторых прочитанных констант рассчитываются номера начальных секторов для каталога и затем для собственно информации (получаются уже известные нам из *разд. 5.6.1* значения 19 и 33). После этого читается начальный сектор каталога и выводится на экран его первая половина.

Примечание

Здесь кроется одна из причин, по которой дискета перед экспериментом должна быть отформатирована: на чистой дискете первые файловые записи обязательно попадут именно в этот сектор; на заполненной дискете может случиться по-другому.

Итак, вы увидите на экране записи каталога, каждая из которых занимает на экране 2 строки по 16 байт. Введите номер по порядку для той, где написано имя интересующего вас файла (он обязательно будет нечетным!) В ответ на это программа для данного файла вычислит адрес поля со смещением 26 (см. табл. 7.2) и извлечет двухбайтовый номер первого кластера в переменную `nc`. Далее печатается имя выбранного файла и происходит считывание сектора 1, т. е. самого первого сектора FAT (опять-таки, если диск был предварительно отформатирован, для экспериментов хватит одного сектора из девяти). Вычислив и запомнив номер самого первого сектора файла в переменную `s0`, программа переходит к извлечению номеров кластеров (или секторов, что в нашем случае почти то же самое, поскольку кластер равен сектору). Цепочка завершается, когда считывается оговоренное заранее фиктивное значение FFF, обозначающее последний кластер файла.

Примечание

Уже отмечалось, что кластеры нумеруются со значения 2; вот почему в формуле для `s0` вычитается двойка.

После печати расшифрованной цепочки номеров для контроля считывается и выводится содержимое самого первого сектора файла. Если файл был текстовым, то проверить правильность выбора несложно.

После того, как вы разберетесь с размещением на дискете одного файла, можно при желании продолжить эксперименты, дописав еще пару коротких файлов. Особый интерес представляют действия системы, если первый из записанных файлов удалить, а затем записать еще один новый файл. Изучите, как поступит система в этом случае, самостоятельно.

Хочется обратить внимание читателей на то, что FAT-система организации хранения файлов является далеко не единственной. Мы не будем рассматривать логическое устройство файловой структуры других ОС. Ограничимся

лишь тем, что упомянем системы Windows NT и 2000, в которых наряду с FAT может использоваться альтернативная система NTFS (NT File System).

Выводы. Таким образом, мы отчетливо видели, как устроена логическая структура дискеты. Информация хранится в виде отдельных файлов, о каждом из которых делается специальная запись в каталог. Каталог дополняется FAT-таблицей, позволяющей размещать части файла на диске произвольным образом. Последнее обстоятельство позволяет более экономно использовать дисковое пространство при многократном удалении файлов.

7.10.5. Слои программного обеспечения

Можно ли как-то обнаружить существование слоев программного обеспечения, о которых рассказывалось в *разд. 7.3*? Те, кто внимательно проделывал все эксперименты, фактически уже знает ответ и может привести довольно яркий пример. В самом деле, сопоставьте наши эксперименты из *разд. 5.6.1 и 5.6.2* с предыдущим *(разд. 7.10.4)*. Вы увидите три разных уровня доступа к информации на диске через разные слои ПО: физический (через подпрограммы BIOS, хранящиеся в ПЗУ), логический (через функции BIOS операционной системы) и файловый, характерный для пользовательского способа доступа к информации.

В данном эксперименте рассмотрим еще один пример взаимодействия с внешним устройством через разные слои ПО.

Цель эксперимента. Принять символы с клавиатуры, используя разные уровни программного обеспечения.

Для проведения эксперимента используем несложную программу, приведенную в листинге 7.3.

Листинг 7.3. Программа, считывающая введенные символы с клавиатуры

```
PROGRAM keyboard;
VAR rh,rl:byte; c:char;
BEGIN WRITELN;
      WRITELN('ROM BIOS');
      ASM mov ah,0
          int 16h
          mov rh,ah
          mov rl,al
      END;
      WRITELN('коды: ',rh,' ',rl,' символ: ',chr(rl));
      WRITELN('DOS');
      REPEAT
```

```
      ASM mov ah,7 {без эхо-печати}
          int 21h
          mov rh,ah
          mov rl,al
      END;
      WRITELN('коды: ',rh,' ',rl,' символ: ',chr(rl));
      UNTIL rl<>0;
END.
```

Программа состоит из двух, на первый взгляд, очень похожих половинок, которые, тем не менее, имеют множество различий. Первая часть реализует доступ к клавиатуре через ROM BIOS, вызывая функцию 0 прерывания 16_{16}. Вторая делает то же самое, но уже на более высоком уровне — через операционную систему (функция 7 прерывания 19_{16}). В обоих случаях младший байт результата помещается в переменную `rl`, а старший — в `rh`, а затем они выводятся на экран.

Запустим программу и дважды нажмем одну и ту же клавишу, например, символ "Q". На экране появится:

ROM BIOS

коды: 16 81 символ: Q

DOS

коды: 7 81 символ: Q

В случае ROM BIOS первый код является так называемым *скан-кодом* клавиши, который однозначно определяется ее расположением на клавиатуре и не зависит от состояния каких-либо регистров (убедитесь сами!) Значение второго числа для обоих случаев очевидно — стандартный код ASCII символа. Цифра 7, все время выводимая в качестве первого числа для строки DOS, не зависит вообще ни от чего и при внимательном размышлении оказывается просто номером функции, который мы сами занесли в регистр `ah`.

Опробуйте нашу программу для различных символьных и цифровых клавиш. Обязательно сравните коды цифр основной и дополнительной клавиатур: вы обнаружите, что их скан-коды различны и, следовательно, ПО на уровне BIOS легко может их различить, чего нельзя сказать об уровне DOS. Еще один весьма неожиданный результат получается для функциональных клавиш. В частности, для клавиши <F1> на экране появится следующее:

ROM BIOS

коды: 59 0 символ:

DOS

коды: 7 0 символ:

коды: 7 59 символ: ;

Мы видим, что подобные клавиши возвращают на уровне BIOS код символа 0, зато на уровне DOS они преобразуются в последовательность из двух (!) символов: 0 и номер клавиши.

Примечание

Именно для того, чтобы зафиксировать эту последовательность, во второй половине программы использован цикл `REPEAT`.

Отметим, что на уровне языков программирования чаще всего используется та же самая логика, что и для уровня DOS. Операторы ввода типа `READLN` или `READKEY` хорошо известны, поэтому здесь мы их рассматривать не будем.

Вывод. В разных слоях программного обеспечения компьютер может работать с внешними устройствами по-разному. Чем ниже уровень ПО, тем ближе он к hardware и, соответственно, тем больше технических деталей имеется. На самом верхнем — пользовательском — уровне мы не можем непосредственно наблюдать даже такую фундаментальную вещь, как коды символов.

7.10.6. Проверка антивирусного ПО

Автор почти уверен, что на вашем компьютере установлено то или иное антивирусное программное обеспечение. Задавали ли вы себе вопрос — а оно работает? Конечно, если вы часто переносите информацию на дискетах или компакт-дисках, то, наверняка, хоть одного вируса обнаружили и без колебаний ответите утвердительно. А если нет? Тогда предлагаю провести следующий абсолютно безопасный эксперимент.

Цель эксперимента. Убедиться в работоспособности антивирусной программы.

Сам эксперимент будет простым, но из-за большой практической важности вопроса стоит сказать несколько слов о том, что такое компьютерный вирус и как он может распространяться. Вполне допускаю, что многие читатели уже интересовались данной тематикой и знакомы с ней; в таком случае можно сразу перейти к выполнению эксперимента.

Некоторые сведения о вирусах и их распространении. Итак, в дополнение к приведенной в *разд. 7.2* классификации, существует еще один весьма специфический вид программного обеспечения, который способен самостоятельно устанавливаться на компьютер без всякого на то разрешения. Речь идет о компьютерных вирусах.

Компьютерный вирус — это специально написанная программа, одной из целей работы которой является скрытое автоматическое создание тем или иным способом собственных копий. Чаще всего помимо "функции размножения" в подобное, с позволения сказать, ПО закладываются некоторые деструктивные функции, например, удаление определенных (или даже всех!) файлов или "противодействие" нормальной работе пользователя. Вирусы могут, в частности, сильно замедлять работу машины, самыми разнообразными способами препятствовать нормальному функционированию программ-

ного обеспечения, наконец, просто "надоедать" пользователю, требуя, допустим, время от времени набирать с клавиатуры фразу типа "хочу печенья". Поскольку злонамеренная фантазия плохо поддается нормальной человеческой логике, перечислить все возможные вредительства не представляется возможным.

Пользователям полезно понимать общие принципы распространения вирусов, поскольку подобные знания могут во многих случаях предотвратить попадание вируса в компьютер. Аналогичным образом санитарное просвещение в медицине способно существенно затормозить распространение инфекционной болезни.

Вирус не является чем-то самостоятельным, и поэтому обычно внедряется в другую компьютерную программу. В частности, он может изменять содержимое загрузочного сектора и тем самым модифицировать процесс загрузки в необходимом ему направлении. Он может также дописывать свой код в исполняемые файлы, и тогда "зараженная" программа помимо своих первоначальных действий незаметно начинает выполнять некоторые дополнительные, происходящие от вируса операции. В последнее время широкое распространение получили новые типы вирусов — так называемые макровирусы, которые могут внедряться в различные документы Microsoft Office: они базируются на развитых средствах внутреннего языка программирования этого пакета — VBA (Visual Basic for Applications). Большой скоростью распространения обладают почтовые вирусы, спрятанные во всевозможных современных динамических "украшательствах" писем E-mail или в исполняемых файлах, помещаемых в приложение к письму; в последнем случае их часто маскируют под фотографии или другие привлекательные материалы (вспомните материал о вирусах, приведенный в *разд. 1.1*). Многие почтовые вирусы способны находить на компьютере адресную книгу и производить автоматическую "саморассылку" по ее адресам. Стандартный текст, обычно используемый при этом, по необходимости неопределенный и нехарактерный для личной переписки, так что внимательный пользователь вполне может заподозрить неладное, прочитав такое письмо от имени своего друга.

Следует особо подчеркнуть, что вирус по определению не может содержаться в данных: ASCII-текстах, графических или звуковых файлах, поскольку он по своей сути является программой и требует исполнения своего кода.

Примечание

Файлы MS Word и электронные таблицы, порождаемые Excel, не являются данными в полном смысле слова. Они имеют сложную структуру и помимо собственно данных могут содержать закодированные действия — макросы. А еще существуют особые ASCII-файлы с командами ОС, например, AUTOEXEC.BAT, куда в принципе вирус может поместить вызовы тех или иных действий, однако они будут заметны.

Рассмотрим возможный механизм распространения вируса на примере макровирусов MS Word. Допустим, некий неуравновешенный и обиженный на весь мир субъект (или недоучившийся студент, пытаясь доказать окружающим свою значимость) пишет вирус. Он добавляет его к тексту интригующего содержания и либо передает его нескольким людям, либо выкладывает в Интернет или рассылает его по электронной почте. Ничего не подозревающий пользователь открывает текст в MS Word и читает. В момент открытия макровирус автоматически запускается и начинает свое незаметное "черное дело". Чаще всего он находит файл со стандартным шаблоном текста NORMAL.DOT и добавляет к нему код своей программы. В результате NORMAL.DOT оказывается "зараженным" и теперь любой текст, набранный в редакторе, автоматически будет содержать копию макровируса. Дальнейший механизм понятен: эти тексты откроют на другой машине, она тоже станет источником распространения вируса и т. д.

Чтобы как-то предотвратить лавинообразный процесс расползания вирусов, существует специальное антивирусное программное обеспечение. Оно содержит обширные базы данных с образцами существующих вирусов и пытается их найти в хранящихся на компьютере файлах. Дополнительно к этому современные антивирусные программы способны находить некоторые типичные конструкции, которые обычно используют вирусы (заметим в скобках, что и обычное ПО может содержать подобные конструкции, поэтому иногда подобный эвристический метод поиска неизвестных вирусов может подать "ложную тревогу").

Примечание

Следует отдавать себе отчет в том, что никакой антивирус не дает 100 %-ной гарантии обнаружения всех вирусов, хотя бы потому, что имеющийся в вашем компьютере вирус мог появиться после последнего обновления базы данных или просто по каким-то причинам, пока неизвестным авторам антивирусного ПО.

Теперь мы, наконец, подошли к непосредственной проблеме нашего эксперимента: можно ли как-то убедиться в работоспособности антивирусного ПО, естественно, не подвергая свой компьютер излишнему риску. Идея состоит в имитации специальным образом "заражения" вирусом. Задача создания безопасной программы, на которую бы реагировало антивирусное программное обеспечение, имеет различные решения. В нашем эксперименте предлагается воспользоваться известным свободно распространяемым тестом антивирусных программ, предложенных Европейским институтом антивирусных исследований EICAR. Данный тест обладает несколькими достоинствами:

- предельно малый размер — всего 68 байт;
- содержит только текстовые символы, поэтому легко может передаваться внутри любого текстового описания;

- необходимый для тестирования исполняемый файл создается очень просто;
- является настоящим исполняемым файлом формата .COM и при запуске выдает на экран сообщение "EICAR-STANDARD-ANTIVIRUS-TEST-FILE".

Для создания теста необходимо набрать или скопировать в любой текстовый редактор следующую строку:

```
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

а затем сохранить ее в формате обычного текстового файла. Размер файла будет всего 68 байт. Читателям книги не придется проделывать кропотливый перенос этой "абракадабры" в свой компьютер — готовый файл с именем EICAR.com находится на прилагаемом к книге CD-диске.

После запуска антивирусного программного обеспечения мы увидим картинку, аналогичную приведенной на рис. 7.11 для программы Dr.Web.

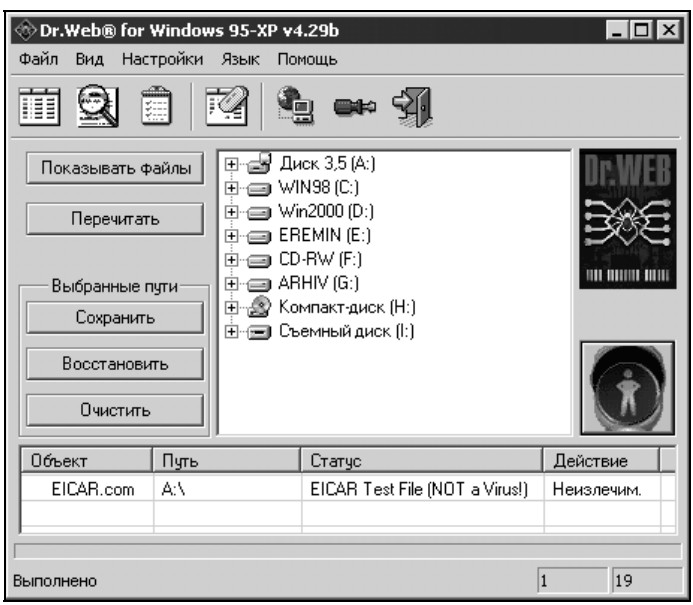
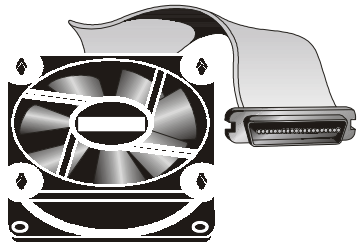


Рис. 7.11. Обнаружение имитации вируса программой Dr.Web

Для тех, кто заинтересовался описанным тестом, приведем несколько ссылок на интернет-источники [И-23, И-3, И-13].

Вывод. Проведенный эксперимент подтверждает работоспособность антивирусного ПО на нашем компьютере.

ЛЕКЦИЯ 8



Учебные модели компьютера

Как мы отчетливо видели на протяжении всего нашего курса, существуют объективные трудности изучения основных принципов устройства и работы вычислительной техники, ибо последняя все усложняется и изменения происходят с такой скоростью, что содержанию образования необычайно трудно поспеть за ними. Следовательно, необходим тщательный отбор наиболее существенного и фундаментального материала, который "стареет" не так быстро. Нужна и определенная популяризация изложения, чтобы не отпугнуть тех, кто только начинает изучать основы устройства вычислительной техники.

Один из подходов, позволяющих решить проблему, состоит в рассмотрении некоторой обобщенной модели ЭВМ, "очищенной" от второстепенных технических деталей.

Автор давно занимается проблемой учебных моделей ЭВМ и даже по мере своих скромных сил поддерживает в Интернете небольшой сайт по данному вопросу [И-18]. Учебным моделям ЭВМ и возможностям их использования при изучении устройства компьютера и посвящается последняя в рамках нашего курса лекция.

8.1. Реальный компьютер или модель?

Итак, допустим, мы решили ознакомиться с фундаментальными основами вычислительной техники. Имеется, по крайней мере, два альтернативных варианта изучения данного вопроса. Первый — попытаться разобраться в работе того реально существующего компьютера, которым вы пользуетесь, и второй — рассмотреть некоторую упрощенную его модель. Как обычно, каждый способ имеет свои достоинства и свои недостатки. Кроме того, реальные компьютеры и их виртуальные модели необычайно разнообразны, поэтому однозначно ответить на вопрос, какой из подходов лучше, довольно сложно.

Знакомство с реальным компьютером дает практически полезные знания, которые непосредственно могут быть использованы в будущем. С одной только неприятной оговоркой — если изученный компьютер "доживет" до этого будущего. С другой стороны, модель, которая содержит наиболее существенные черты, легче понять и можно сконцентрировать свое внимание именно на этих чертах. Но и здесь не обходится без трудностей: она должна быть достаточно хорошей, чтобы отражать по-настоящему фундаментальные свойства, которые мало меняются со временем. Кроме того, с обсуждаемой точки зрения важно, что проектирующие реальный компьютер инженеры не особенно склонны заботиться об удобстве его изучения, тогда как модель специально создается для этой цели.

Суммируя все вышесказанное, можно сделать вывод, что для целей изучения основ наилучшим образом подходят четко и логично спроектированные реальные компьютеры или тщательно разработанные модели, причем последние, по-видимому, имеют некоторое преимущество.

Проанализируем теперь оба варианта подробнее с позиций настоящего момента.

Начнем с того, что в 1985 году, когда в школе был введен новый курс ОИВТ, в нашей системе образования были наиболее распространены компьютеры с архитектурой PDP-11 (см. [54]): ДВК, БК, несколько позднее — КУВТ УКНЦ (дополнительные материалы о составе этого семейства можно найти в [И-26]). Архитектура семейства PDP фактически стала классической (в Интернете находится много страниц, наполненных ностальгической грустью об ее ушедших достоинствах; см. ссылки по адресу [И-16]), поэтому неудивительно, что в первых учебниках А. П. Ершова за основу были взяты именно такие машины. Но времена менялись, и на смену пришло новое семейство — IBM PC. В нашей стране описываемый процесс произошел достаточно безболезненно, поскольку отечественная техника имела невысокую надежность на фоне впечатляющего технического качества импортных машин (скажем, дискеты с одной ДВК не всегда читались на другой; впрочем, купленные в последнее время трехдюймовые дискеты живо напоминают нам эти трудности, казалось бы, давно прошедших времен).

С точки зрения простоты изучения, семейство IBM PC на базе процессоров Intel существенно хуже, чем "старый добрый" PDP. По-видимому, одной из главных причин является стремление сохранить полную совместимость с самыми первыми машинами семейства. Уже с трудом запускаются в новейших версиях Windows приложения MS-DOS, часть принтеров принципиально неспособны печатать в этой уходящей системе русские тексты, но Intel Pentium по-прежнему поддерживает все команды и режимы работы самого первого процессора этой серии 8086. Разумеется, многие свойства предыдущих моделей, мягко говоря, не являют собой вершину логической прозрачности. Достаточно вспомнить сегментный метод адресации памяти *(см. разд. 4.3.2)*, введенный как временное решение для 16-разрядных машин;

сейчас он уже абсолютно не нужен, но все еще "занимает место" в архитектуре процессоров.

Кроме того, изделия Intel принадлежат к классу CISC-процессоров, т. е. система их команд сознательно спроектирована сложно. И если в PDP все регистры использовались в машинных инструкциях единообразно, то в процессорах Intel каждый регистр имеет строго определенное назначение, из-за чего приходится помнить довольно много деталей.

Таким образом, с точки зрения интересов нашей сегодняшней лекции, реальные компьютеры на процессоре Intel не являются хорошим объектом для изучения фундаментальных принципов. Причинами являются их чрезмерная для начинающих сложность и огромное, накопившееся за много поколений процессоров, количество технических нюансов.

Обратимся теперь ко второму подходу, связанному с учебными моделями ЭВМ. Большинство школьных учебников использует самые разнообразные модели, так что главная трудность состоит не в отсутствии моделей как таковых, а скорее, наоборот, в их многообразии и, к сожалению, не всегда хорошем соответствии принципам работы современных компьютеров.

Поскольку с реальным процессором на современном этапе нам, похоже, не очень повезло, имеет смысл рассмотреть подробнее имеющиеся в литературе модели для изучения устройства ЭВМ. Краткому обзору учебных моделей вычислительных машин и посвящен следующий раздел. Автор надеется, что знакомство с разнообразными по устройству и по системе команд моделями (например, с различным количеством адресов в команде или всевозможными способами доступа к данным) будет определенным дополнением к материалу нашего курса и расширит представления читателей о возможных принципах построения ЭВМ.

8.2. Обзор существующих учебных моделей

Идея использования в учебном процессе вместо реальной ЭВМ ее упрощенной модели появилась достаточно давно. Например, в журнале "Математика в школе" эта проблема довольно подробно обсуждалась уже в 1977 году [4]. Тем не менее изучение ЭВМ в то время проводилось только в некоторых специализированных школах. Введение курса основ информатики и вычислительной техники во все средние учебные заведения сделало знакомство школьников с этим вопросом поистине массовым, что дало рассматриваемому методу новый толчок для развития. Поэтому наш обзор естественно начать именно с этого периода.

8.2.1. Модели ЭВМ в учебниках информатики

Как уже указывалось в предыдущем разделе, в первом учебнике информатики, предложенном А. П. Ершовым [31, 32], за основу было взято изучение

наиболее подходящего для этой цели реального процессора PDP. Этот же подход, но в значительно меньшем объеме, был сохранен в учебнике [50]. Не замедлило появиться и программное обеспечение в поддержку этого методического направления: на распространенной тогда в вузах японской ПЭВМ Yamaha получил широкое распространение имитатор PDP [2]. Он мог наглядно показывать содержимое памяти и регистров виртуального процессора PDP (на самом деле, Yamaha была собрана на базе микропроцессора Z80) и позволял вводить и отлаживать небольшие программы, написанные на своеобразном "русском Ассемблере". В систему команд имитатора были отобраны наиболее важные инструкции, предназначенные для обработки целых чисел. Фактически имитатор PDP представлял собой некоторую учебную модель, но только устройство и поведение этой модели достаточно близко соответствовало реальному процессору. Попутно заметим, что в то время автор также принял активное участие в разработке методики изучения основ вычислительной техники на базе процессора PDP [22, 23].

Последующие учебники начали постепенно отходить от описанной линии преподавания. Одна из первых попыток была сделана в [46], где предложено рассмотреть работу процессора "на упрощенной модели ЭВМ". Правда, самой этой модели отведено всего четыре странички, но этого вполне достаточно, чтобы увидеть, что это уже не PDP.

Новый шаг от изучения реальной ЭВМ к учебной модели был сделан коллективом свердловских авторов [11]. Они предложили законченную модель учебной ЭВМ, практически реализованную в виде учебной программы [11, 5]. Модель носила название Кроха и представляла собой классическую трехадресную ЭВМ, соответствующую первому или второму поколению. Благодаря "крошечному" размеру ОЗУ — 8 ячеек, адрес в такой ЭВМ занимал всего 3 бита, а для всей команды, состоящей из кода операции и трех адресов, требовалось 12 бит.

Кроха Мозель умела работать только с целыми положительными 12-разрядными числами: от 0 до 4095. Ее система команд содержала всего 8 инструкций: перепись, 4 арифметических операции, условные переходы по результатам сравнения на "равно" и "больше" и, наконец, останов, совмещенный с выводом на экран. Экран дисплея учебной ЭВМ состоял из трех строк, в каждую из которых выводилось одно число. Клавиатура у "Крохи" отсутствовала, а ввод осуществляется путем прямого редактирования содержимого памяти. Неявно подразумевалось, что это делается с некоторого пульта управления аналогично тому, как это было у первых вычислительных машин. Видимо, на этом же пульте должны были находиться и управляющие кнопки **Пуск**, **Стоп** и др.

Несмотря на кажущуюся простоту модели, на ней можно показать довольно много. Например, если, не изменяя системы команд, увеличить память до 16 ячеек, то удается даже реализовать простейший демонстрационный компилятор языка Паскаль [24].

Дальнейшее развитие модели учебной ЭВМ предложено в [64, 70]. Разработанный в этой учебной литературе компьютер Нейман тоже имел трехадресную систему команд, близкую к той, что была у Крохи. Как и Кроха, Нейман обрабатывал только целые числа, но зато они могли быть отрицательными и диапазон их допустимых значений заметно шире: от — 2 147 483 648 до 2 147 483 647.

Существенное преимущество Неймана перед Крохой в том, что объем его ОЗУ значительно больше — 256 байт, которые организованы в виде 64 машинных слов по 32 разряда каждое. Как показывает опыт, такого количества вполне достаточно для решения практически любой учебной задачи. Важной особенностью модели также служило наличие специальной ячейки для ввода и вывода данных: через нее происходил ввод информации с клавиатуры и вывод на индикатор. Отметим, что такая система обмена информацией с внешними устройствами скорее напоминает работу микрокалькулятора, чем реальной ЭВМ. Тем не менее налицо определенное продвижение вперед в моделировании работы внешних устройств по сравнению с Крохой.

Важной методической особенностью компьютера Нейман является возможность демонстрации нескольких уровней программирования. Сначала это было программирование в машинных кодах и на языке Ассемблер [64]. Позднее авторы добавили возможность использования языка программирования Паскаль [70]. Интересно, что все три уровня тесно связаны, т. к. применена некогда популярная на машинах семейства ДВК технология трансляции с Паскаля в текст на Ассемблере и уже из него в машинный код.

Следующей моделью учебной ЭВМ стала Малютка [71], первоначально предложенная для школ с углубленным изучением информатики. По сравнению с предыдущими моделями, она имела развитую систему из 26 команд. Помимо уже упоминавшихся выше арифметических действий и переходов, Малютка способна выполнять логические операции, сдвиги и даже выдачу звукового сигнала, "тональность которого определяется содержимым сумматора". Малютка — это одноадресная ЭВМ, что является для учебных ЭВМ принципиально новым моментом. В процессоре Малютки есть специальный внутренний регистр — сумматор, в котором выполняются все операции. Именно в сумматор извлекаются данные, там выполняется обработка информации, и из него результаты записываются в память. Иначе говоря, вся система команд построена на работе с сумматором. Для нашего курса подобная организация системы команд представляет большой интерес и поэтому в *разд. 8.6.3* мы познакомимся с ней несколько подробнее.

Как и Нейман, Малютка имела 256 ячеек ОЗУ, только они 12-разрядные. Видимо, это следует признать менее удачным, чем байтовая структура памяти, принятая в Неймане. Адреса данных, как и у Неймана, входили непосредственно в состав команд; никаких других способов адресации предусмотрено не было.

Малютка предназначалась для обработки числовой информации, причем числа могли быть как целые, так и дробные. Наличие вещественных чисел является важной отличительной чертой модели. "Подгоняя" способы хранения данных под свою модель, авторы сделали оба типа чисел 12-битовыми, но форма их представления очень похожа на то, как это выглядит в "настоящей" ЭВМ.

При рассмотрении Малютки была поставлена задача об обработке массива данных на модельной ЭВМ. Правда способ, предлагаемый для этого, довольно нестандартный: программа переформировывала свои собственные команды прямо в ОЗУ. Выбранный метод для Малютки, конечно, единственно возможный, но он далек от того, что используется на практике (кстати говоря, попробуйте формировать программу, находящуюся в ПЗУ, где информацию нельзя изменять!).

Заявленная возможность использования звука в учебнике практически не описана.

По поводу внешних устройств ЭВМ Малютка в тексте учебника также сказано довольно мало. Внимательное чтение позволяет заметить, что для вывода результатов имеется специальное табло, которое дает "возможность сохранять... много целых значений сумматора в одном из трех форматов — в формате целых чисел, в формате дробных чисел и в двоичном (шестнадцатеричном) формате". О существовании клавиатуры удается догадаться только по наличию специальной команды ввода с клавиатуры; ее действие в учебнике не описывается и в примерах не обсуждается.

Из методических достоинств описания модели в [71] следует обязательно отметить подробное разъяснение методики написания программы на языке низкого уровня. Реализован также язык Ассемблер, а по поводу языков высокого уровня замечено, что их "можно было бы при желании научить... понимать" и Малютку.

8.2.2. Учебный компьютер E97

Таким образом, проведенный выше анализ эволюции имеющихся в отечественной учебной литературе моделей ЭВМ позволяет обнаружить и достаточно четко проследить их развитие и совершенствование. По-видимому, переход от конкретного процессора к его упрощенной модели является правильным с точки зрения удобства изучения основ ВТ. Тем более, что, как мы видели в *разд. 8.1*, господствующие в настоящее время повсюду процессоры фирмы Intel как объект изучения достаточно неудобны, что также подтверждает выбор в пользу разумной учебной модели.

С другой стороны, рассмотренные модели нуждаются в значительном усовершенствовании, поскольку они так мало похожи на современный Pentium!

В качестве наиболее существенных отличий можно выделить следующие:

1. Учебные ЭВМ (кроме имитатора PDP, повторяющего "настоящий" процессор) никак не учитывают основополагающие принципы устройства современного четвертого поколения ЭВМ. Наоборот, все они навивают ностальгические воспоминания о "старых добрых временах" машин 1—2 поколения.

2. Ни одна из моделей не работает с нечисловыми данными; нельзя продемонстрировать даже обработку такого важного и широко распространенного вида информации, как текст.

3. Во всех современных ЭВМ давно используется байтовая организация памяти, но в учебных ЭВМ это либо совсем не отражается, либо отражается лишь частично, как, например, в компьютере Нейман. Во всяком случае, везде команды и все обрабатываемые данные для простоты модели предполагаются одинаковой длины, что не соответствует действительности, начиная с третьего поколения.

4. Только в описании модели Малютка делается попытка (хотя, по-видимому, и не совсем удачная) показать, как процессор может обрабатывать массивы информации — а в этом состоит одно из главных предназначений ЭВМ! Данная ограниченность моделей является принципиальной: ни в одну из них не заложены методы адресации, позволяющие организовать доступ к последовательно расположенным однородным данным.

5. Ни одна из описанных моделей не позволяет по-настоящему объяснить, как процессор работает с внешними устройствами. Имеющиеся табло и достаточно неопределенные устройства ввода слишком условны и к тому же имеют очень мало общего с периферийными устройствами современного компьютера.

6. Существующие модели не позволяют реализовать одно из важнейших достижений программирования — переход с возвратом, т. е. вызов подпрограммы. Иными словами, мы "теряем" способ реализации одной из фундаментальных алгоритмических структур. По-видимому, это во многом связано с необходимостью изучать "секреты" работы стекового механизма организации памяти. В то же время, мы с вами уже видели, что принципы стека оказываются лишь ненамного сложнее разборки и сборки детской пирамидки *(см. разд. 4.3.3)*.

Подчеркнем, что главная цель написанного выше состоит совсем не в том, чтобы показать, как плохи существующие модели, а указать пути их необходимого усложнения. В свете сформулированных положений, автор в 1997 году поставил задачу создать свободную от описанных выше ограничений учебную модель современного компьютера. Она была кратко названа E97, где буква "скромно" символизирует разработчика модели, а цифры обозначают год ее создания. Первое подробное описание учебной модели E97 было

опубликовано малым тиражом в книге [25]. Позднее часть этого описания была включена в вышедший в центральном издательстве учебник [58]. Соответствующий раздел имеется также в задачнике [59].

В состав учебного микрокомпьютера Е97 входят следующие устройства: центральный процессор, память двух видов (ОЗУ и ПЗУ), а также два наиболее важных внешних устройства — клавиатура для ввода информации и дисплей для вывода. Последние, как принято в современных ЭВМ, подключены через порты ввода/вывода.

Главным блоком компьютера служит 16-разрядный процессор, способный работать как с двухбайтовыми словами, так и с отдельными байтами, т. е. с данными разной длины. Познакомившись с тем, как Е97 обрабатывает разные типы информации, можно впоследствии легко обобщить логику "один байт или много" на случай большей разрядности процессора.

В процессоре имеются внутренние регистры, при помощи которых реализован метод косвенной адресации к памяти, подробно описанный в *разд. 4.3.2.*

Большинство инструкций Е97 имеют одинаковую структуру, приведенную на рис. 8.1.

Рис. 8.1. Базовая структура команд учебного компьютера Е97

Изображенная на рисунке 16-разрядная команда состоит из четырех одинаковых частей, каждая из которых кодируется самостоятельной шестнадцатеричной цифрой. Модификатор **MOD** задает некоторые особенности исполнения инструкции (хорошим примером может служить условие, по которому происходит условный переход). Далее следуют код операции **OP** и два операнда **S** ("источник") и **D** ("приемник"). Уточним, что в качестве операнда **S** может использоваться содержимое регистра, значение из ячейки памяти, на которое показывает регистр, а также ячейка со входящим в инструкцию адресом или константа. Второй операнд **D**, в который принято записывать результат, имеет те же самые возможности за исключением константы.

Конкретные примеры инструкций Е97 будут приведены позднее при практической реализации программ *(см. разд. 8.6.2)*.

В Е97 существует память двух видов — ОЗУ и ПЗУ. В первой хранится текущая информация по решаемой задаче, причем она может как считываться, так и записываться. Во второй, предназначенной только для чтения, содержатся разработанные при проектировании ЭВМ подпрограммы наиболее часто используемых действий, среди которых важное место занимают алгоритмы обмена с внешними устройствами. Вся память имеет байтовую организацию, свойственную современным компьютерам.

Имеет смысл рассмотреть ПЗУ подробнее, поскольку в учебных ЭВМ оно появляется впервые. Наличие в модели ПЗУ дает целый ряд преимуществ, важнейшими из которых являются следующие.

Во-первых, можно показать, что в современных вычислительных машинах не всякое элементарное действие реализуется в виде отдельной инструкции процессора: многие из них выполняются по программам. В E97, например, это перевод целого числа из двоичной системы в десятичную. Наличие таких подпрограмм освобождает модель от необходимости вводить в систему команд учебного процессора разные нестандартные и несуществующие в реальных процессорах инструкции типа "вывод числа на экран в десятичной форме" и ей подобные.

Во-вторых, наличие ПЗУ позволяет не конкретизировать алгоритм работы с внешними устройствами на уровне системы команд процессора, а перенести их в ПЗУ. Именно так делается и в настоящих компьютерах, где имеется специальное ПЗУ, которое называется BIOS *(см. разд. 4.1.2)*. Помимо "реализма" такой подход позволяет на начальных этапах обучения не вникать в детали ввода/вывода, а просто вызывать соответствующие подпрограммы ПЗУ.

Кроме описанного выше, использование ПЗУ приводит к появлению в модели дополнительных преимуществ методического плана. Оформив, как это сделано в E97, ПЗУ в виде текстового файла, мы получаем замечательную возможность легко модифицировать работу нашей модели. По мнению автора, стремление разработать свою собственную полезную стандартную подпрограмму и затем добавить ее в ПЗУ, может служить мощным стимулом для изучения деталей работы компьютера. Наконец, подробно прокомментированный текстовой файл с ПЗУ может служить хорошим наглядным примером того, как нужно составлять программы.

Завершая обсуждение модели, укажем на еще одну ее интересную особенность (точнее говоря, речь идет о программной реализации). В E97 предусмотрена возможность выбирать команды модели по желанию пользователя: последний может сам скомпоновать систему инструкций виртуального компьютера, включая в нее любое подмножество из стандартного набора.

Некоторые подробности работы с моделью E97 будут продемонстрированы в ходе выполнения после лекции традиционных экспериментов.

8.2.3. Зарубежные модели

В зарубежной образовательной литературе также часто используется прием объяснения основ вычислительной техники на базе упрощенной модели. В качестве примера назову, например, недавно вышедшую книгу [8]. Много моделей предлагается на страницах Интернета, но, к сожалению, большинство из них оформлены на английском языке и из-за чего их применение в российских условиях затруднено.

Наиболее известной и подробно проработанной теоретической моделью, без сомнения, является MIX Дональда Кнута [49]. Наверное, нет человека, интересующегося компьютерными вычислениями, которому был бы неизвестен фундаментальный многотомный труд Д. Кнута "Искусство программирования на ЭВМ". В нем проведено всестороннее исследование практически всех наиболее важных вычислительных алгоритмов. В качестве примера достаточно упомянуть, что в одном из разделов книги обсуждается влияние временны́х характеристик движения магнитной ленты на эффективность сортировки больших массивов информации. Такой детальный подход к исследованию вычислительных задач невозможен без анализа времени исполнения машинных операций, а значит, без рассмотрения конкретной системы команд ЭВМ. Д. Кнут в своей книге предложил условную вычислительную машину собственной конструкции, которую назвал MIX (слово "mix" переводится с английского как "смесь"). MIX является виртуальной машиной, но она очень похожа на наиболее распространенные ЭВМ того времени. С лукавым юмором Кнут заявляет, что MIX есть римская запись числа 1009, получающегося как средний номер моделей всех машин, которые он проанализировал при создании своей модели [49]:

(360+650+709+7070+U3+SS80+1107+1604+G20+B220+S2000+920+
+601+H800+PDP4+II)/16

Несмотря на то, что модель MIX содержит минимальное количество технических деталей, за четыре десятилетия она все-таки устарела. И вот недавно Кнут решил ее модернизировать и заменить на более современную, которую он назвал MMIX. Сам автор рекомендует читать сокращение как "эм-микс", причем первая буква M "скромно" обозначает "millennium" — т. е. новое тысячелетие. Здесь снова есть столь любимый Кнутом шутливый подтекст: если название считать состоящим из римских цифр, то из них образуется число 2009, которое автор предполагает годом выпуска окончательной редакции своего многотомника.

Хотя MMIX создан Д. Кнутом для изучения эффективности работы различных численных методов, тем не менее, это не единственно возможное его применение. Благодаря хорошему соответствию между моделью и реальными процессорами (Кнут упоминает, что при разработке он во многом ориентировался на процессоры фирмы AMD), MMIX может быть использован для знакомства с основными принципами устройства и функционирования современных компьютеров. К сожалению, даже продаваемое сейчас в магазинах новое издание книги "Искусство программирования на ЭВМ" пока по-прежнему базируется на MIX. А описание MMIX существует лишь в виде книги на английском языке [82] и обновляемых авторских материалов, размещаемых на интернет-сайте Д. Кнута [И-25]. Некоторое предварительное описание на русском языке, опубликованное в газете "Информатика" [30], едва ли можно считать исчерпывающей информацией.

В связи с вышеизложенным, учитывая отсутствие доступной информации на русском языке об этой интересной и масштабной разработке (справедливо претендующей на роль классической), в следующем разделе модель MMIX будет рассмотрена более подробно.

8.2.4. MMIX Д. Кнута — RISC-процессор тысячелетия

64-разрядный RISC-компьютер MMIX способен обрабатывать широкий спектр самых разнообразных данных: однобайтовые символы ASCII и более современные двухбайтовые коды Unicode, целые числа различной длины (вплоть до 8 байт) как без знака, так и со знаком и, наконец, 64-разрядные вещественные числа.

MMIX имеет трехадресную систему команд, построенную по традиционной для RISC-машин схеме: все команды работают исключительно с регистрами процессора, а для записи данных в ОЗУ и считывания их оттуда используются две специализированные инструкции. MMIX имеет полный набор целочисленных и вещественных арифметических операций, 16 логических инструкций, широкий ассортимент условных и безусловных переходов, сдвиги и прочие традиционные для реальных процессоров операции. Всего система команд процессора насчитывает 256 инструкций, каждая из которых имеет длину 4 байта.

Адресное пространство машины имеет 64 бита, что позволяет адресовать огромное количество памяти. Доступ к ней байтовый, с использованием косвенного регистрового и индексного методов. Обширный набор регистров (до 256 одновременно) позволяет реализовывать очень сложные алгоритмы обработки информации.

Примечание

Регистры в MMIX образуют единый регистровый файл, подобно тому, как это делается в современных RISC-процессорах. Такая организация рассматривает все множество регистров как некоторую динамическую информационную структуру, причем нумерация регистров в ней не является жесткой и легко может изменяться программным путем. Наличие подобного механизма переключения регистров дает возможность осуществить быстрый и эффективный вызов подпрограмм с параметрами, что является необычайно важным удобством для реализации языков программирования высокого уровня. Описание принципов работы регистрового файла хотя и представляет определенный интерес для специалистов, тем не менее, достаточно сложно и потому не входит в задачи нашей книги. Наоборот, при первом знакомстве можно без особого ущерба считать, что нумерация регистров фиксирована и неизменна со временем.

Приведенные характеристики дают нам некоторое первоначальное представление о MMIX. Сразу становится очевидным, что это очень подробная модель, полное освоение которой явно выходит за рамки данной книги. В то же время, некоторые ее особенности стоит рассмотреть хотя бы для

того, чтобы после лекции успешно выполнить эксперимент, запланированный в *разд. 8.6.1.*

Начнем изучение устройства MMIX с форматов команд. Они довольно разнообразны, но большая их часть сводится к трем изображенным на рис. 8.2.

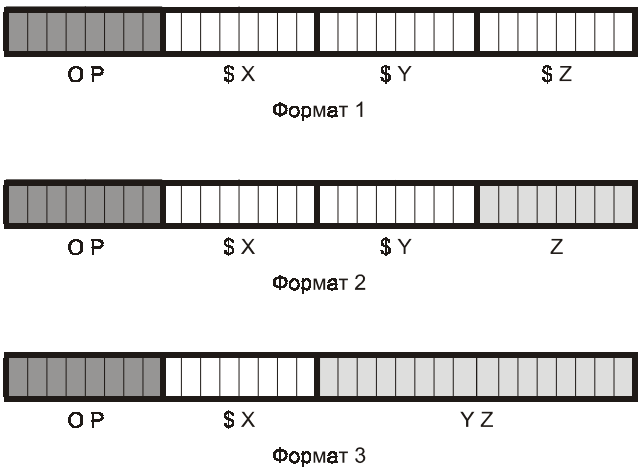


Рис. 8.2. Основные форматы команд компьютера MMIX

Формат 1 содержит однобайтовый код операции **OP** и три таких же по размеру поля с номерами регистров **$X**, **$Y** и **$Z**. Выполнение любой операции строится по схеме, общепринятой для всех языков программирования:

```
$X := $Y <операция> $Z
```

где вместо обозначения `<операция>` может стоять любое арифметическое, логическое или иное двухместное действие, которое определяется кодом `OP`. Приведем несколько примеров:

- 20 02 04 08 (`ADD $2, $4, $8`) — сложить регистры `$4` и `$8`, результат поместить в `$2`;
- C8 06 04 05 (`AND $6, $4, $5`) — выполнить логическую операцию "И" над регистрами `$4` и `$5`, результат поместить в `$6`;
- AC 07 08 09 (`STO $7, $8, $9`) — сохранить значение регистра `$7` в ячейку памяти с адресом, равным сумме значений регистров `$8` и `$9`.

Формат 2 очень похож на предыдущий с той лишь разницей, что вместо второго операнда в нем используется целая беззнаковая константа `Z` от 0 до 255. Инструкции работы с константами всегда имеют код на единицу больший, чем соответствующие операции с регистрами. Например:

21 02 04 08 (`ADDI $2, $4, 8`) — к содержимому регистра `$4` прибавить 8, результат поместить в `$2` (сравните с первым примером для предыдущего формата).

Казалось бы, описанных выше форматов должно хватать. Тем не менее имеется еще одна очень важная и весьма специфическая проблема, связанная с необходимостью задавать требуемые значения регистров. Принципиальная трудность состоит в том, что любой регистр MMIX содержит 64 двоичных разряда, а команда содержит ровно половину. Кроме того, не забывайте, что 16 из этих 32 разрядов заняты кодом операции и номером результирующего регистра; таким образом, под собственно константу, которую можно поместить в инструкцию, остается всего 16 бит! Один из возможных выходов в сложившейся ситуации состоит в том, чтобы регистр разделить на четыре части по 16 бит, каждая из которых заполняется своей собственной командой. Нетрудно сообразить, что при таком подходе для занесения в регистр полного 64-битового значения придется выполнить четыре инструкции. Каждая из них имеет специализированный **Формат 3**, который состоит из следующих частей: кода операции **OP**, в котором помимо всего прочего содержится указание на необходимую "четверть" регистра, номера последнего и, наконец, 16-разрядной константы, которую необходимо занести (на рис. 8.2 она обозначена как **YZ**).

Например, команда с кодом E1 03 FFFF (`SETMH $3, FFFF`) устанавливает значение регистра `$3` равным 0000 FFFF 0000 0000, т. е. в определенную пару байт заносится указанная константа FFFF, а все остальные байты обнуляются.

Помимо обсуждавшихся выше инструкций для задания значений регистров, **Формат 3** также используется в переходах. Здесь мы не будем рассматривать эти интересные операции; желающие получить представление о том, как реализуются переходы в MMIX, могут обратиться к уже упоминавшейся ранее статье [30].

Как уже отмечалось выше, все команды MMIX работают только с регистрами. Исключение составляют две особые инструкции — запись и чтение ячеек памяти. Они тоже имеют **Форматы 1** или **2**, но работают по другой схеме. Для того чтобы, например, записать в ОЗУ требуемую информацию, необходимо предварительно в одном из свободных регистров задать адрес ячейки, а затем выполнить команду обмена с памятью, содержащую ссылку на номер последнего регистра. Подобный механизм адресации данных, когда адрес интересующей ячейки памяти хранится не в самой команде, а в регистре процессора, а тот, в свою очередь, указывается в качестве операнда, называется методом косвенной адресации *(см. разд. 4.3.2)*. Именно такой метод лежит в основе работы всех современных микропроцессоров.

Однако косвенным методом возможности адресации данных в MMIX не исчерпываются. Адрес ячейки памяти на самом деле вычисляется несколько более сложным способом: он является суммой двух величин — вычисляется как содержимое одного из регистров плюс содержимое другого или константа. Понять сказанное позволяет рис. 8.3, на котором приводится схема записи содержимого регистра в ОЗУ.

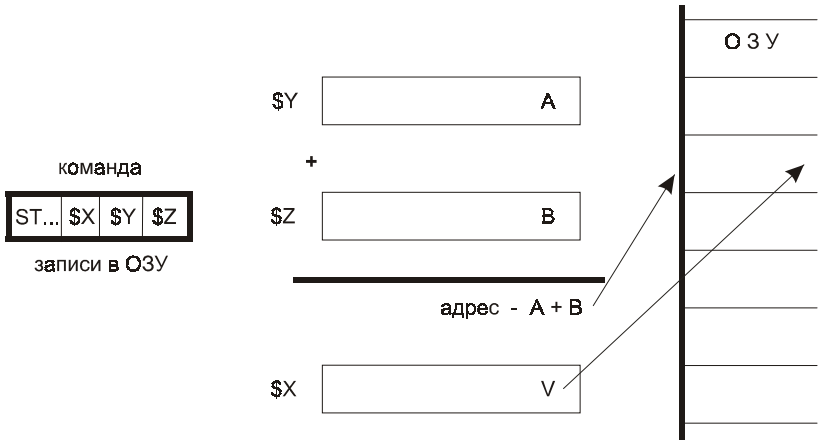


Рис. 8.3. Адресация к памяти в компьютере MMIX

Из рисунка отчетливо видно, что адрес ячейки ОЗУ, в которую производится запись данных, есть сумма второго и третьего операндов инструкции **$Y** и **$Z**. Именно в этот адрес памяти и происходит запись значения из регистра **$X**.

Примечание

Для записи в память одного, двух, четырех и восьми байт регистра в MMIX имеются разные операции: `STB`, `STW`, `STT` и `STO` соответственно. Именно по этой причине на рис. 8.3 мнемоника команды после `ST` содержит вместо последней буквы многоточие.

Рассмотрим для иллюстрации две инструкции записи полного 64-разрядного содержимого регистра `$1` в ОЗУ:

- AC 06 07 08 (`STO $6, $7, $8`) — сохраняет содержимое регистра `$6` в 8 последовательных байт начиная с адреса, равного сумме значений регистров `$7` и `$8`;
- AD 06 07 08 (`STOI $6, $7, 8`) — единственное отличие данной команды от предыдущей состоит в том, что начальный адрес ОЗУ равен сумме значения регистра `$7` и константы `8`.

Нетрудно сообразить, что если в последнем случае положить константу равной нулю, то получится метод косвенной адресации "в чистом виде", когда адрес ОЗУ определяется только одним регистром. Подчеркнем также, что представление адреса операнда в виде суммы двух значений необычайно удобно для доступа к последовательно расположенным данным — массивам, играющим в современных языках программирования весьма существенную роль.

Читатель вправе потребовать от нас конкретного примера использования всех этих знаний. Просим подождать совсем чуть-чуть до проведения самого первого "любопытного эксперимента".

Из данного раздела лекции определенно следует сделать вывод о том, что для начинающих модель MMIX чрезмерно подробна. Поэтому для наших ознакомительных целей вполне достаточно упрощенного (адаптированного) варианта. Именно такая программная реализация, написанная автором книги для общеобразовательных целей, и приведена на CD-диске. Чтобы не путать ее с полной моделью ЭВМ Кнута, для краткости она имеет собственное название — E-MMI (rEduced MMIx) [И-15]. В следующем далее кратком сравнении учебных ЭВМ присутствуют как полная модель MMIX, так и ее упрощенная модификация E-MMI.

8.3. Сравнение учебных моделей

Для того чтобы обобщить информацию о возможностях рассмотренных в нашей лекции учебных моделей ЭВМ, соберем наиболее важные сведения о них в единую таблицу (табл. 8.1). В качестве таких сведений возьмем виды обрабатываемых моделью данных, характеристики системы команд, объем ОЗУ и состав устройств ввода/вывода. Наконец, завершают таблицу данные о составе программного обеспечения, которым оснащены модели.

Таблица 8.1. *Основные характеристики учебных ЭВМ*

	Кроха	**Нейман**	**Малютка**	**E97**	**MMIX**	**E-MMI**
Данные:						
Разрядность (бит)	12	32	12	8/16	8/16/32 /64	8/16/32/64
Целые положительные числа	+	+	+	+	+	+
Целые отрицательные числа	–	+	+	+	+	+
Вещественные числа	–	–	+	–	+	+
Символы	–	–	–	+	+	+
Графика *	–	–	–	+	+	
Работа со звуком	–	–	+	–	–	–
Команды:						
Количество команд	8	10	26	27	256	120
Выбор команд	–	–	–	+	–	–

Таблица 8.1 *(окончание)*

	Кроха	Нейман	Малютка	E97	MMIX	E-MMI
Команды *(прод.)*:						
Количество адресов в команде	3	3	1	2	3	3
раЗрядность (бит)	12	32	12	16/32/48	32	32
Устройства:						
Объем ОЗУ	8 ячеек	256-4 байт	256 ячеек	256 байт	до 2^{63} байт	2 блока по 1024 байт
Объем ПЗУ (байт)	–	–	–	384	–	1024
Устройства ввода	–	Клавиатура	Клавиатура	Клавиатура	**	Клавиатура
Устройства вывода	Дисплей (3 строки)	Индикатор	Дисплей	Дисплей	**	Дисплей
Программное обеспечение:						
Монитор	–	–	–	+	+	+
Ассемблер	–	+	+	–	+	–
Паскаль	+	+	–	+	–	+

* Имеются специальные версии программ с графикой; базовые версии ее не поддерживают.

** "Официальный" имитатор MMIX имеет специальную систему обращений ко внешним устройствам, напоминающую принятую в операционных системах для обращения к клавиатуре, дисплею и дисковым файлам.

Большинство приведенных в таблице данных понятны или обсуждались в тексте лекции ранее. Отметим только последние строки, которые отражают возможность использования различных учебных моделей при изучении принципов работы программного обеспечения. Данные возможности пока используются мало и вполне могут быть расширены, например, добавлением простейших программ сжатия информации и т. п.

Приведенные в табл. 8.1 данные также демонстрируют постепенный рост возможностей учебных моделей, что, очевидно, связано с объективным увеличением сложности устройства вычислительной техники, а также расширением видов обрабатываемой информации.

8.4. Выводы

Итак, проведенное в лекции рассмотрение учебных моделей ЭВМ позволяет сделать следующие выводы.

- Имеется, по крайней мере, два альтернативных варианта изучения основ ВТ. Первый — попытаться разобраться в работе реально существующего компьютера, и второй — рассмотреть некоторую упрощенную его модель. Как обычно, каждый способ имеет свои достоинства и свои недостатки.
- Для образовательных целей наилучшим образом подходят четко и логично спроектированные реальные компьютеры или тщательно разработанные модели, причем последние, по-видимому, имеют некоторое преимущество. Господствующие ныне процессоры семейства Intel являются не самым удобным объектом для изучения фундаментальных принципов.
- В учебной литературе имеется большое количество моделей ЭВМ. Автор надеется, что рассмотренные в лекции материалы и рекомендации помогут подобрать модель для большинства целей образования и самообразования.
- По мнению автора, возможности моделей значительно шире, чем принято обычно считать. Например, на базе учебных ЭВМ можно изучать алгоритмы, основы программного обеспечения и другие вопросы, связанные с процессами обработки информации на компьютере. Тщательно проработанная модель ЭВМ может являться одной из основ учебного курса.

8.5. Вопросы для осмысления

1. Попробуйте, опираясь на материал курса и свои собственные представления, назвать несколько положений об устройстве вычислительной техники, которые оставались и остаются справедливыми в течение длительного времени. Как много подобных положений вам удалось найти?
2. Согласны ли вы с тем, что многие вопросы информатики тесно связаны и даже опираются на фундаментальные положения об устройстве ВТ? Приведите несколько аргументов, подтверждающих вашу точку зрения.
3. Какие аргументы можно привести за и против, обсуждая выбор между реальной ЭВМ и учебной моделью? Какие из них, по-вашему, более убедительны?
4. Сравните описанные в лекции учебные модели ЭВМ. Каковы достоинства и недостатки каждой из них?
5. Существенна ли для учебной модели возможность обработки вещественных чисел? Текстов? Графической и других видов информации?
6. Предположим, компьютер вывел на экран некоторое целое число, которое хранится в ОЗУ. Какие действия при этом потребовалось выполнить?

По каким причинам (постарайтесь назвать несколько) в систему команд процессора не входит вывод на экран значения числа?

7. Насколько полезно в учебной модели компьютера ПЗУ? Какую информацию туда стоит поместить?
8. Какие характеристики в табл. 8.1 являются наиболее существенными, а какие не имеют особого значения?

8.6. Любопытные эксперименты

Как обычно, в нашем курсе раздел с таким названием содержит ряд упражнений, иллюстрирующих теоретический материал. В данном случае, проводя эксперименты, читатели смогут убедиться в том, что с помощью учебных моделей компьютера можно узнать о настоящих компьютерах много полезных вещей.

Первые три эксперимента, описанные в *разд. 8.6.1—8.6.3* объединены единой тематикой. В них ставится цель показать, как происходят расчеты по одной и той же формуле на ЭВМ с различным устройством системы команд: трех-, двух- и одноадресных. Возможно, перед проведением экспериментов стоит вернуться и повторно перечитать *разд. 3.3*, где обсуждаются базовые принципы построения системы инструкций процессоров.

В качестве трехадресной ЭВМ мы используем модель MMIX Д. Кнута, описанную в *разд. 8.2.4*. Для изучения двухадресной ЭВМ рассмотрим учебный компьютер E97 *(см. разд. 8.2.2)*, а при знакомстве с одноадресной машиной обратимся к учебной ЭВМ Малютка, о которой шла речь в *разд. 8.2.1*.

Для всех трех экспериментов выберем единую не слишком сложную формулу:

$$z = ax + by$$

Подчеркнем, что в ходе вычислений по приведенной выше формуле образуется промежуточная величина ax, которую необходимо сохранить до момента получения произведения by. Такая ситуация при реализации вычисления по формулам на ЭВМ является вполне типичной.

Наконец, мы везде будем предполагать, что данные хранятся в ячейках ОЗУ. Такое предположение не просто наилучшим образом соответствует реальной ситуации, но и попутно позволит нам познакомиться на практике с различными методами адресации памяти, используемыми в вычислительной технике.

8.6.1. Вычисления на трехадресной ЭВМ

Начнем с наиболее простой для понимания трехадресной ЭВМ. Как уже говорилось выше, в качестве модели воспользуемся предложенным Д. Кнутом RISC-компьютером MMIX.

Цель эксперимента. Реализовать вычисление по заданной формуле на трехадресной машине.

Несколько возможных форматов инструкций MMIX подробно были описаны в *разд. 8.2.4*. Как следует из изложенного там материала, трехадресная команда MMIX чаще всего состоит из четырех однобайтовых частей: кода операции, номера регистра для записи результата, регистра первого операнда и либо регистра со вторым операндом, либо беззнаковой константы от 0 до 255, непосредственно включенной в код команды. Примеры конкретных инструкций RISC-процессора MMIX приведены в табл. 8.2. Поясним, что обозначения `$X`, `$Y`, `$Z` определяют произвольные регистры, а `Z` — константу.

Таблица 8.2. *Примеры некоторых инструкций MMIX (полную систему команд см. в [82] или [И-25])*

Код	Мнемоника	Описание	Пример
8D $X $Y Z	`LDOI $X, $Y, Z`	Содержимое 8-байтового числа извлекается из адреса `$Y+Z`	8D 02 00 08 — число из ячейки `$0+8` считывается в `$2`
AD $X $Y Z	`STOI $X, $Y, Z`	Регистр `$X` сохранить в память по адресу `$Y+Z`	AD 02 00 20 — сохранить содержимое `$2` в ячейку `$0+20`
20 $X $Y $Z	`ADD $X, $Y, $Z`	`$Y + $Z ==> $X`	18 03 01 02 — сложить регистры `$1` и `$2`; результат занести в `$3`
18 $X $Y $Z	`MUL $X, $Y, $Z`	`$Y * $Z ==> $X`	20 02 02 03 — перемножить регистры `$2` и `$3`; результат занести в `$2`
E3 $X YZ	`SETL $X, YZ`	Константу `YZ ==> $X`	E3 00 0030 — занести константу 30 в регистр `$0`
00 00 00 00	`TRAP 0`	Выход в ОС	

Первые две команды используют в качестве второго операнда константу и их коды нечетны, а две последующие оперируют с регистром и у них коды операций, напротив, четные. Особый формат имеют две последние операции в таблице. Команда `SETL` с кодом E3 устанавливает указанный в ней регистр равным двухбайтовому значению `YZ`, непосредственно входящему в состав команды; старшие 6 байт регистра автоматически заполняются нулями. Наконец, команда `TRAP` с нулевыми параметрами реализует завершение программы и возвращает управление запускавшей ее операционной системе.

Разобравшись в приведенных выше примерах, мы легко сможем понять программу вычисления по интересующей нас формуле (табл. 8.3).

Таблица 8.3. *Программа вычисления по заданной формуле*

Адрес	Код	Мнемоника	Действие	Комментарий
0	E3 00 0030	`SETL $0, 0030`	`$0 <== 30`	Начальный адрес в `$0`
4	8D 01 00 00	`LDOI $1, $0, 0`	`$1 <== ($0+0)`	Считать `a` из ОЗУ в `$1`
8	8D 02 00 08	`LDOI $2, $0, 8`	`$2 <== ($0+8)`	Считать `x` в `$2`
C	18 03 01 02	`MUL $3, $1, $2`	`$3 <== $1 * $2`	`a * x` в `$3`
10	8D 01 00 10	`LDOI $1, $0, 10`	`$1 <== ($0+10)`	Считать `b` в `$1`
14	8D 02 00 18	`LDOI $2, $0, 18`	`$2 <== ($0+18)`	Считать `y` в `$2`
18	18 02 01 02	`MUL $2, $1, $2`	`$2 <== $1 * $2`	`b * y` в `$2`
1C	20 02 02 03	`ADD $2, $2, $3`	`$2 <== $2 + $3`	`ax + by` в `$2`
20	AD 02 00 20	`STOI $2, $0, 20`	`($0+20) <== $2`	Сохранить `z` из `$2` в ОЗУ
24	00000000	`TRAP 0`	Выход в ОС	Завершение задачи

При разборе программы полезно помнить о расположении используемых в задаче переменных в памяти MMIX, которое приводится в табл. 8.4.

Таблица 8.4. *Расположение переменных в памяти MMIX*

Адрес	Код	Число	Комментарий
30	0000 0000 0000 0002	2	`a`
38	0000 0000 0000 0003	3	`x`
40	0000 0000 0000 0004	4	`b`
48	0000 0000 0000 0005	5	`y`
50			`z = ax + by`

Примечание

Обратите внимание на тот факт, что команды в MMIX занимают 4 байта, а числа — 8. Именно поэтому свести их в единую таблицу затруднительно.

Остается ввести нашу программу в имитатор MMIX, расположенный на прилагаемом к книге CD-диске. Однако прежде чем начинать работать, полезно выяснить, как с программой E-MMI "общаться".

Предлагаемая вашему вниманию программа базируется на исторически существовавшем не так давно интерфейсе командной строки. Программы, которые с помощью подобного интерфейса управляли процессом разработки ПО в машинных кодах, называли мониторами (не путать с одноименным устройством компьютера!) *Монитор* — это специальная программа, позволяющая загружать в оперативную память, просматривать, тестировать и изменять любую программу в машинных кодах. Монитор не просто дает возможность запускать находящуюся в памяти отлаживаемую программу, но также позволяет отслеживать каждый шаг ее выполнения, анализировать содержимое ОЗУ, ПЗУ и внутренних регистров процессора. Программы такого типа раньше являлись главным инструментом разработчиков программного обеспечения.

Наиболее известными отладочными программами для 8-разрядных процессоров являлись DDT, SID и ZSID. В дальнейшем с ростом разрядности процессоров параллельно усложнялись и отладочные средства для них. В основном это коснулось интерфейса, который стал многооконным и приобрел широкие возможности экранного редактирования (например, достаточно просто подвести курсор к нужному месту кода на экране и исправить его). В качестве примеров назовем отладчики AFD и TurboDEBUGGER для IBM, великолепную отладочную программу Л. Бараза DBG для компьютера Yamaha. Последняя являлась в свое время одной из самых любимых программ автора данной книги.

Для работы с MMIX тоже требуется монитор. Он составляет с программной моделью учебного микрокомпьютера единое целое и позволяет вводить, редактировать и запускать на вашем реально существующем компьютере IBM программы, написанные для воображаемой микроЭВМ MMIX.

Можно считать, что монитор находится в ПЗУ и что он перехватывает для обработки все возникающие в отлаживаемой программе ошибки и даже момент ее завершения. При "включении" нашего учебного компьютера, т. е. при запуске программы-имитатора, управление также передается монитору и он сразу же готов начать диалог с вами. На первый взгляд это выглядит несколько искусственно и неправдоподобно, но некоторые настоящие ЭВМ работали в точности так. В частности, у машин семейства PDP программа-монитор действительно хранилась в ПЗУ и фактически составляла нижний уровень ПО (кстати, помните наш разговор о слоях ПО в *разд. 7.3*?).

Монитор для MMIX понимает несложную систему директив (их можно было бы называть и командами, но мы будем везде употреблять термин "директивы", чтобы не путать с командами процессора!) Она во многом похожа на реализованную в упоминавшейся ранее профессиональной программе DDT: желающие могут обратиться для сравнения к достаточно известной

книге М. Уэйта и Дж. Ангермейера [77], посвященной операционной системе CP/M. Каждая директива монитора состоит из латинской буквы, указывающей, какое действие необходимо выполнить, и не всегда обязательного шестнадцатеричного параметра. Последний часто может опускаться, а для некоторых директив просто отсутствует.

Примеры директив:

- `D30` — вывести на экран часть содержимого ОЗУ начиная с адреса 30;
- `G` — запуск программы с адреса 0 (адрес подразумевается);
- `Q` — выход из монитора, т. е. завершение работы с программой-имитатором (параметр отсутствует).

Полный список директив монитора приведен в табл. 8.5.

Таблица 8.5. *Полный перечень команд монитора программы E-MMI*

Команда	Параметр	Действие	Пример	Комментарий
`$` ($<регистр>)	Номер регистра	Показать содержимое регистра и изменить его	`$41`	`$` эквивалентно `$0`
`B` (Bytes)	4 или 8	Установка режима команд `S`, `W`, `R`: 4 (инструкции) или 8 байт (данные)	`B4` `B8`	`B` эквивалентно `B4`
`C` (Characters)	Адрес	Ввод текста	`C18`	`C` эквивалентно `C0`
`D` (Dump)	Адрес	Показать данные в виде Hex-кодов и символов	`D18`	`D` эквивалентно `D0`
`F` (Float)	Адрес	Показать данные в виде Hex-кодов и вещественных чисел	`F18`	`F` эквивалентно `F0`
`G` (Go)	Адрес	Исполнить программу	`G14`	`G` эквивалентно `G0`
`I` (Integer)	Адрес	Показать данные в виде Hex-кодов и целых чисел	`I18`	`I` эквивалентно `I0`
`J` (Jump)	Адрес	Вычисление перехода	`J14`	`J` эквивалентно `J0`
`L` (List)	Адрес	Показать программу в виде Hex-кодов и мнемоник	`L14`	`L` эквивалентно `L0`
`M` (Model)	–	Показать информацию о модели	`M`	
`P` (PC)	–	Показать значение программного счетчика и изменить его	`P`	

Таблица 8.5 *(окончание)*

Команда	Параметр	Действие	Пример	Комментарий
Q (Quit)	–	Выход в ОС	Q	
R (Read)	–	Читать файл (.COD — программа, .DAT — данные; см. команду B)	R	Имя файла вводится в диалоге
S (Set)	Адрес	Показать содержимое ячейки памяти и изменить ее (см. также команду B)	S14	S эквивалентно S0
T (Trace)	Число	Выполнить несколько инструкций программы	T5	T эквивалентно T1
W (Write)	Число байт	Сохранить файл (.COD — программа, .DAT — данные; см. команду B)	W40	Имя файла вводится в диалоге
X (extended)	Номер регистра	Показать значение регистров во всех формах	X56	X эквивалентно X0

Примечания

- Все значения вводятся в шестнадцатеричной форме.
- Адреса 8-байтовых данных должны делиться на 8, а адреса команд — на 4.
- Для выхода из режимов S и $ используйте клавишу <Esc>.
- Для прерывания MMIX программы используйте комбинацию клавиш <Ctrl>+<C>.

Наконец мы знаем достаточно, чтобы сесть за компьютер. Запустим программу с виртуальным компьютером MMIX, которая находится на прилагаемом CD-диске в файле a:\8\E-MMI\E-MMI.exe. На экране появится картина, изображенная на рис. 8.4.

Для набора программы введем директиву S и нажмем клавишу <Enter>; в ответ монитор выведет адрес начальной ячейки 0 и ее текущее содержимое 00010203 (при старте программы память MMIX для удобства тестирования заполняется возрастающей последовательностью байтов). Теперь наберем первую команду программы E3000030. После ввода ее последней, восьмой шестнадцатеричной цифры курсор автоматически "перескочит" на очередную строку и выведет содержимое следующей ячейки с номером 4. В результате на экране отобразится такой диалог:

```
4>s
0    0001 0203 E3000030
4    0405 0607
```

(набранные нами символы в тексте отмечены курсивом, все остальное выводит на экран монитор).

Примечание

Обязательно обратите внимание на выведенный монитором в начале первой строки текст, содержащий цифру 4. Эта информация сообщает о том, что при старте монитор находится в режиме программы и ввод ведется по 4 байта. Чуть позднее мы научимся изменять этот режим для перехода ко вводу данных.

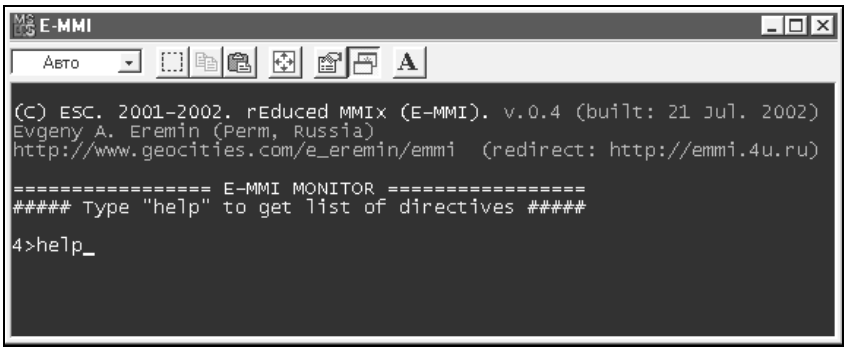


Рис. 8.4. Программная реализация модели E-MMI

Далее аналогичным образом наберите остальные инструкции программы. Будьте внимательны, ибо любая неправильная цифра приведет к получению неверного результата. После завершения ввода последней состоящей из одних нулей команды, нажмите клавишу <Esc> для выхода из режима ввода.

Внимательно проверьте набранные коды. Для исправления замеченных ошибок воспользуйтесь тем же самым режимом `S`, только вместо правильно набранных значений нажимайте клавишу <Enter>, а рядом с неправильными пишите исправленный код.

Нам еще остается ввести данные в ячейки ОЗУ, начиная с адреса 30. Для этого наберем директиву `B8`, чтобы переключиться в восьмибайтовый режим ввода данных, а затем с помощью директивы `S30` попросим монитор вводить числа начиная с ячейки 30. Значения, вообще говоря, могут быть любыми, но для удобства проверки лучше взять небольшие числа. Диалог ввода данных будет выглядеть на экране следующим образом:

```
4>b8
8 bytes is set for W, R and S commands

8>s30
30   3031 3233 3435 3637 2
38   3839 3A3B 3C3D 3E3F 3
```

```
40   4041 4243 4445 4647 4
48   4849 4A4B 4C4D 4E4F 5
50   5051 5253 5455 5657 0
58   5859 5A5B 5C5D 5E5F

8>
```

Таким образом, мы ввели в память MMIX значения переменных `a`, `x`, `b` и `y` `2`, `3`, `4` и `5` соответственно и "на всякий случай" очистили ячейку с номером 50 под результат.

Теперь все готово к запуску с помощью директивы `G`. Выполнив программу, монитор выведет содержимое регистров — нам есть смысл обратить внимание на содержимое регистра `$2`, где находится ответ. Если вы использовали те самые значения, которые указаны выше, результат будет равен $1A_{16}$ или 26_{10}. Остается убедиться, что такой же результат лежит в ячейке ОЗУ с номером 50, для чего достаточно набрать директиву вывода целых чисел `I30`.

Вот, собственно, и все. Конечно, после интерфейса Windows работа с командной строкой несколько утомляет, но, если подойти непредвзято, повторить эксперимент с помощью приведенных выше детальных пояснений не так уж сложно.

Примечание

Существует еще один сокращенный вариант проделать описанный выше эксперимент — не набирать самому программу, а прочитать ее с CD-диска. Для этого сразу после запуска программы наберите директиву `R` и в ответ на запрос программы наберите имя файла E8_1. В данном файле предусмотрена загрузка не только программы, но и данных, так что сразу после ввода можно запустить программу директивой `G`.

Автор не надеется, что абсолютно всем читателям понравится работа с программой E-MMI, но если такое вдруг случится, то вы можете попробовать расписать свою собственную формулу и добиться ее правильного вычисления на ЭВМ.

Разумеется, учебный компьютер E-MMI "умеет" значительно больше, чем то, что потребовалось нам для описанного эксперимента, но обсуждение его возможностей выходит за рамки данной книги. Желающие могут обратиться к подробной статье автора о MMIX, опубликованной в газете "Информатика" [30].

Вывод. Таким образом, мы подробно разобрались, как реализуются вычисления на трехадресной машине. В следующем эксперименте мы проделаем ту же работу на двухадресной машине.

8.6.2. Вычисления на двухадресной ЭВМ

А теперь посмотрим, как выглядят вычисления по нашей формуле

$$z = ax + by$$

на процессоре с двухадресной системой команд. Для этого воспользуемся учебным компьютером E97, программная реализация которого также имеется на прилагаемом к книге CD-диске. Поскольку принципы реализации ПО во многом аналогичны рассмотренной в предшествующем разделе модели, описание эксперимента можно без особого ущерба сделать менее подробным.

Цель эксперимента. Для сравнения с предыдущим экспериментом провести вычисление по той же самой формуле на двухадресной машине.

Как описано в *разд. 8.2.4*, основная часть команд процессора E97 состоит из четырех частей по 4 бита (т. е. по одной шестнадцатеричной цифре) каждая: модификатора, изменяющего некоторые детали исполнения команд, кода операции и двух операндов. В простейшем случае операнды — это регистры процессора, но могут также использоваться ячейки ОЗУ или константы.

При удалении из состава команды одного адреса (по сравнению с трехадресной ЭВМ) возникает проблема указания местоположения результата операции. В E97 принято помещать его вместо второго операнда, так что о сохранности старого значения данного операнда программист должен заботиться самостоятельно.

Примеры инструкций E97 приведены в табл. 8.6. Буквы `S` и `D` в кодах команд обозначают операнды, над которыми производится действие, а `K` — константу.

Таблица 8.6. *Примеры некоторых инструкций учебного компьютера E97 (полную систему команд см. в [25, 58] или [И-18])*

Код	Описание	Пример
01SD	Копировать операнд `S` в `D`	0141 — содержимое ячейки, адрес которой в `R0`, скопировать в `R1`
02SD	Сложить операнды `S` и `D`; результат сохранить в `D`	0221 — сложить содержимое регистров `R2` и `R1`, результат — в `R1`
22KD	К операнду `D` прибавить константу `K`; результат — в `D`	2220 — увеличить содержимое `R0` на 2
05SD	Перемножить операнды `S` и `D`; результат сохранить в `D`	0532 — перемножить регистры `R3` и `R2`, результат — в `R2`
0F00	Останов	

Обязательно обратите внимание, что в тех примерах, где вместо `S` и `D` указаны числа от 0 до 3, процессор будет использовать в качестве данных непо-

средственное значение соответствующих регистров R0–R3, а для чисел 4—7 содержимое регистра будет использоваться как адрес ячейки ОЗУ с необходимыми данными. Так, в приведенной в первой строке табл. 8.6 команде, первый операнд равен 4, что обозначает ячейку памяти, адресуемую через R0. Подобный метод доступа к данным, как мы уже знаем, называется косвенной адресацией *(см. разд. 4.3.2)*.

Еще одной особенностью E97 по сравнению с MMIX является отсутствие возможности индексирования адреса ОЗУ относительного указанного регистра. Как вы, конечно, помните, в программе для MMIX мы просто устанавливали значение регистра $0 на начало блока данных, а все переменные задавали, указывая в качестве константы z необходимое смещение. В E97 при переходе к очередной ячейке данных приходится каждый раз пересчитывать значение регистра: поскольку данные находятся в последовательных ячейках, то регистр R0 при этом достаточно увеличивать на 2: процессор E97 16-разрядный, т. е. размер ячейки составляет 2 байта.

Учитывая все описанные особенности, программа для вычислений по изучаемой формуле примет следующий вид (табл. 8.7).

Таблица 8.7. *Окончательный вид программы вычисления по заданной формуле*

Адрес	Код	Действие	Комментарий
0	01D0	30 ==> R0	Адрес a в R0
2	0030		
4	0141	(R0) ==> R1	Считать a из ОЗУ в R1
6	2220	2 + R0 ==> R0	Адрес x
8	0142	(R0) ==> R2	Считать x в R2
A	0521	R2 * R1 ==> R1	a * x в R1
C	2220	2 + R0 ==> R0	Адрес b
E	0142	(R0) ==> R2	Считать b в R2
10	2220	2 + R0 ==> R0	Адрес y
12	0143	(R0) ==> R3	Считать y в R3
14	0532	R3 * R2 ==> R2	b * y в R2
16	0221	R2 + R1 ==> R1	ax + by в R1
18	2220	2 + R0 ==> R0	Адрес z
1A	0114	R1 ==> (R0)	Сохранить z из R1 в ОЗУ

Таблица 8.7 *(окончание)*

Адрес	Код	Действие	Комментарий
1C	0F00	Стоп	Завершение задачи
...			
30	0002	2	a
32	0003	3	x
34	0004	4	b
36	0005	5	y
38			z = ax + by

Поскольку команды и целые числа в E97 имеют одинаковую длину, они сведены в общую таблицу.

Вероятно, вы захотите убедиться в практической работоспособности написанной программы. Тогда обратитесь к каталогу 8\e97 прилагаемого к книге CD-диска, в котором содержатся две программные реализации учебного компьютера, которые отличаются только способом вывода на экран. Файл E97.exe является традиционной программой в среде MS-DOS, а E97W.exe создает для своего вывода окно в традиционном стиле Windows (тем не менее, во всем остальном это точно такая же программа). По техническим соображениям воспроизводить в книге копию экрана в среде MS-DOS менее удобно, поэтому на всех рисунках приведен именно второй вариант программы. Однако во всех остальных отношениях обе версии практически равноценны и вы можете воспользоваться любой из них.

Примечание

Строго говоря, имеется еще одна очень интересная и удобная в работе реализация учебного компьютера E97, выполненная М. М. Паршиным. Ее можно найти в Интернете по ссылке [И-11].

Итак, вы запустили программу и начинаете с ней работать. Принципы организации взаимодействия с ней полностью аналогичны программе E-MMI, с которой вы познакомились в предыдущем эксперименте. Более того, даже большинство директив монитора совпадают: для ввода программы используется директива `S`, для просмотра содержимого памяти — `D`, а для запуска программы — `G` (полный список директив монитора и их описание приведены в файле 8\e97\E97MON.txt на прилагаемом CD-диске). Так что процесс набора будет практически полностью аналогичен тому, как вы поступали в предыдущем эксперименте, и даже проще, поскольку в E97 отсутствуют различные режимы ввода программы и данных, а значит, и директивы их переключения.

Протокол работы программы после завершения ее набора (еще раз напомним, что для этой цели используется директива S, а для выхода — клавиша <Esc>) приведен на рис. 8.5.

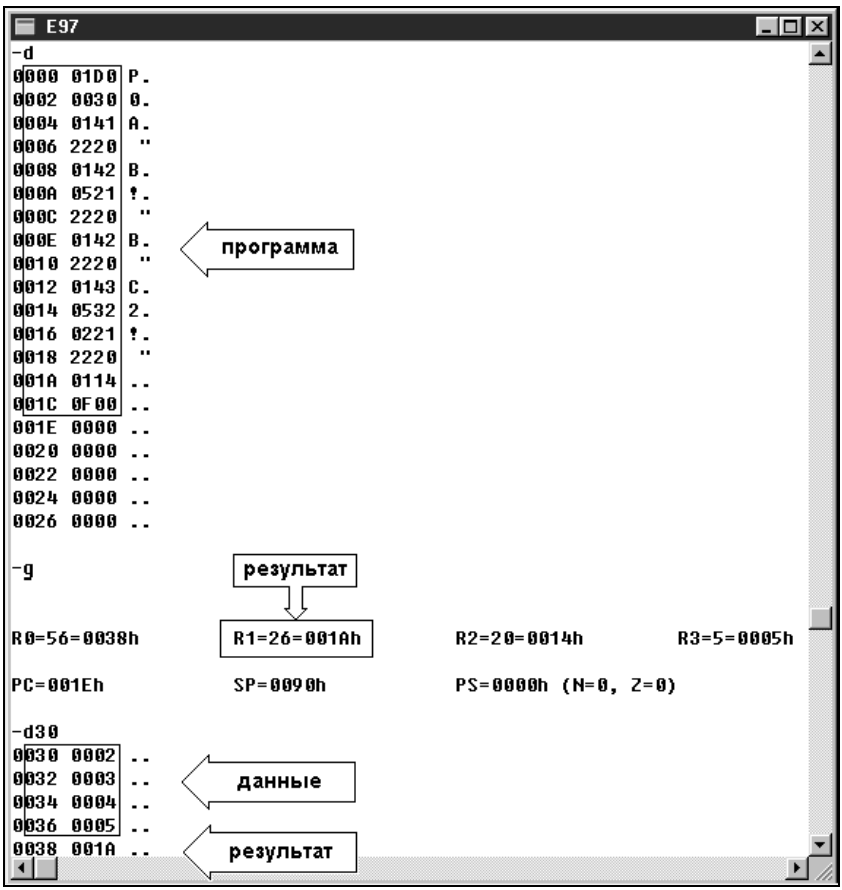


Рис. 8.5. Вычисления по формуле на учебном компьютере E97

Примечание

Как и в эксперименте с E-MMI, вы можете не набирать программу и данные, а просто загрузить файл E8_2 с помощью директивы R.

Для удобства разбора рис. 8.5 некоторые наиболее важные коды выделены с помощью прямоугольных рамок, а специальные нанесенные на рисунок стрелки показывают области программы, данных и результат. Имея перед глазами столь информативный рисунок, читатели легко смогут разобраться в столбиках цифр, выводимых программой.

Вывод. Итак, мы реализовали вычисления по той же самой формуле на двухадресной машине. Осталось посмотреть, как это будет выглядеть на одноадресной машине.

8.6.3. Вычисления на одноадресной ЭВМ

Займемся последним экспериментом из этой серии: разберем, как вычисляется наше выражение на одноадресной машине. Автору известна только одна учебная одноадресная модель — Малютка, предложенная и подробно описанная в учебнике А. И. Сенокосова и А. Г. Гейна [71]. К сожалению, ее программной реализации нет на CD-диске — согласно существующим правилам, автор книги может свободно публиковать только разработанное им самим ПО. Тем не менее программа Малютка бесплатно распространяется в Интернете [И-14] и "скачать" эту небольшую программку по указанной ссылке для читателей не составит особого труда.

Цель эксперимента. Произвести вычисления по формуле на одноадресной модели Малютка.

Как и любая реальная машина, каждая учебная модель ЭВМ тоже имеет свои особенности. Малютка, разумеется, не является исключением из правила. Будучи одноадресной ЭВМ, она во всех своих операциях активно использует особый регистр — сумматор (о роли сумматора мы говорили в *разд. 3.3.3*). Другой особенностью модели является то, что она все обрабатываемые данные хранит только в ОЗУ, и адреса ячеек непосредственно входят в команду. Последнее возможно только потому, что память Малютки невелика — всего 256 ячеек и, следовательно, двух шестнадцатеричных цифр вполне достаточно для представления полного адреса.

Любая команда ЭВМ Малютка состоит из 12-ти двоичных разрядов (обрабатываемые ей данные тоже). Таким образом, для записи машинной инструкции требуется всего 3 шестнадцатеричных цифры. Первая обозначает код операции, а две последующие, как вы уже, наверняка, догадались, адрес используемой ячейки ОЗУ. Примеры некоторых инструкций Малютки показаны в табл. 8.8.

Таблица 8.8. *Примеры некоторых инструкций ЭВМ Малютка (полную систему команд см. в [71])*

Код	Мнемоника	Описание	Пример
0NN	`LDA (NN)`	Содержимое ячейки памяти с номером `NN ==> CM`	`0 09` — содержимое ячейки `09` считывается в `CM`
1NN	`STA (NN)`	`CM ==>` ячейку памяти с номером `NN`	`1 0D` — сохранить содержимое сумматора в ячейку `0D`

Таблица 8.8 *(окончание)*

Код	Мнемоника	Описание	Пример
ANN	ADD (NN)	CM + ячейка NN ==> CM	A 0D — прибавить к CM целое число из ячейки 0D
BNN	MULT (NN)	CM * ячейку NN ==> CM	B 0A — CM умножить на целое число из ячейки 0A
F00	HLT	Останов	

Из этой таблицы отчетливо видно, что для осуществления вычислений один операнд необходимо предварительно извлечь в сумматор, и только потом между ним и вторым операндом из ОЗУ можно выполнить требуемое действие; наконец, результат операции, полученный в сумматоре, при необходимости может быть сохранен в памяти отдельной инструкцией.

Программная реализация Малютки выполнена на других принципах, нежели те, с которыми мы имели дело в первых двух экспериментах. Поэтому прежде чем писать программу, нам необходимо с этими принципами познакомиться.

В отличие от рассмотренного ранее ПО, выполненного в виде программы-монитора и оперирующего непосредственно цифровыми кодами, Малютка представляет собой реализацию языка Ассемблер. Ассемблер является существенным шагом вперед по сравнению с монитором, поскольку допускает использование буквенных обозначений — так называемых *идентификаторов*. Все элементы команды в таком языке являются идентификаторами. В частности, вместо числового кода операции используется его мнемонический эквивалент: для инструкции считывания из ОЗУ вместо цифры 1 пишется lda, а вместо кода операции сложения A — add. Но код операции далеко не главное. Гораздо более существенным является замена конкретных адресов ОЗУ идентификаторами. Дело в том, что при таком подходе из программы полностью исчезает "привязка" к конкретным адресам памяти и ее становится гораздо легче изменять. Но как же тогда, спросите вы, процессор сможет исполнить такую текстовую программу, если он понимает единственный язык — язык цифровых кодов? Кажущееся противоречие разрешается просто: текст на Ассемблере обрабатывается специальной системной программой-транслятором, которая преобразует этот текст в привычную процессору числовую форму. Подчеркнем, что сделать подобный перевод достаточно просто, поскольку каждой строке на Ассемблере соответствует одна машинная команда; фактически достаточно произвести замену определенных сочетаний символов соответствующими двоичными кодами.

После данного пояснения становится понятным, как выглядит программа наших вычислений (табл. 8.9).

Таблица 8.9. *Программа вычислений по заданной формуле на одноадресной ЭВМ*

Метка	Мнемоника	Адрес	Код	Действие	Комментарий
	lda (a)	1	009	(9) ==> см	Считать a из ОЗУ
	mult(x)	2	B0A	см * (A) ==> см	a * x
	sta (z)	3	10D	см ==> (D)	Сохранить в z
	lda (b)	4	00B	(B) ==> см	считать b из ОЗУ
	mult (y)	5	B0C	см * (C) ==> см	b * y
	add (z)	6	A0D	см + (D) ==> см	+ ax
	sta (z)	7	10D	см ==> (D)	Сохранить в z
	hlt	8	F00	стоп	
a:	dw 2	9	002	2	a
x:	dw 3	A	003	3	x
b:	dw 4	B	004	4	b
y:	dw 5	C	005	5	y
z:	dw 0	D			z = ax + by

Дополнительно поясним, что шестнадцатеричной цифрой (старшая цифра всех используемых в задаче адресов ОЗУ равна нулю, поэтому она не указана) в скобках в столбце "Действие" обозначено содержимое соответствующих ячеек Малютки. При этом конкретные числовые значения ячеек и коды команд являются результатом трансляции текстовой программы. Для их получения уже требуется садиться за компьютер и работать с ПО Малютка.

К счастью, интерфейс программы довольно простой и привычный, во многом сродни Турбо Паскалю. После запуска программы мы попадаем в обыкновенный текстовый редактор и легко набираем текст программы (рис. 8.6). Также можно, используя пункт меню **Файл**, просто загрузить готовый файл с именем E8_3, находящийся на CD-диске в каталоге с именем 8. Затем запускаем программу на исполнение (перед этим она автоматически будет оттранслирована), нажав клавишу <F9>.

Результаты трансляции, ради которых во многом и производился данный эксперимент, посмотрим, нажав клавишу <F5> (рис. 8.7).

Дополнительная разметка, нанесенная на рис. 8.7, позволяет легко найти ту информацию, которую мы уже видели в таблице.

Мы в третий раз, теперь уже на одноадресной машине, убедились, что если в формулу $z = ax + by$ подставить значения $a = 2$, $b = 3$, $c = 4$ и $d = 5$, то ЭВМ с любой системой команд получит ответ $1A_{16}$ или 26_{10}.

Рис. 8.6. Программа на Ассемблере для одноадресной ЭВМ Малютка

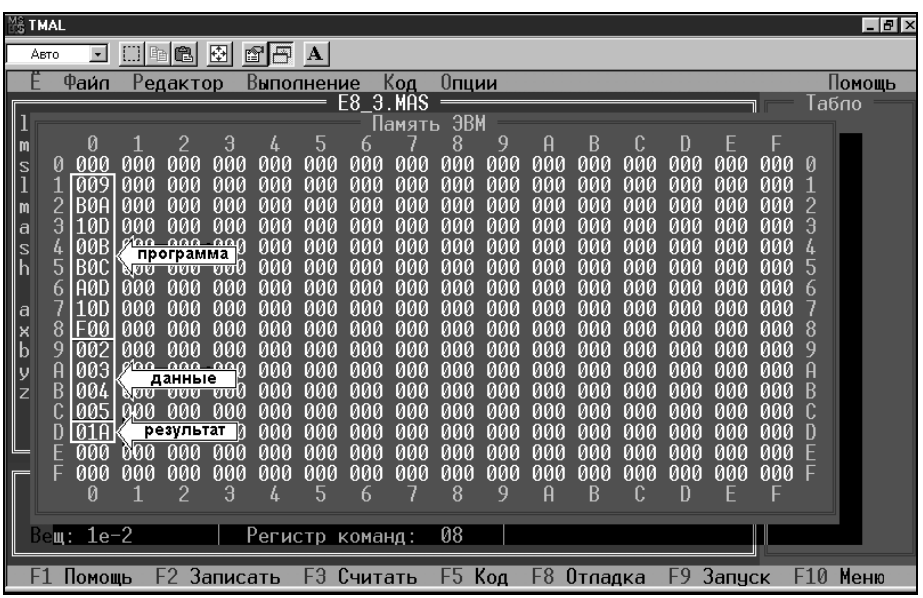


Рис. 8.7. Результат трансляции программы на Ассемблере для ЭВМ Малютка

Выводы. Итак, мы разобрались, как осуществляют вычисления трех-, двух- и одноадресные машины. Особо отметим, что в ходе экспериментов мы рассмотрели несколько методов адресации данных в памяти ЭВМ. Сделано это было отнюдь не специально, а потому, что в учебных моделях ради простоты набор методов адресации ограничен. В реальном современном компьютере, как правило, представлены все эти методы одновременно — а значит, наша не совсем обычная экскурсия в мир моделей была не напрасной.

8.6.4. Как ЭВМ принимает решения

Те, кто не пожалел времени и проделал три предыдущих эксперимента, теперь хорошо представляют себе, как происходят вычисления внутри компьютера (кстати, а вы не забыли, что английское слово computer в своем изначальном значении и есть "вычислитель"?) Но для завершения картины не хватает некоторых очень существенных деталей, а именно: как компьютер выбирает из нескольких вариантов нужный или как он организует многократные вычисления по одним и тем же формулам. Ответам на эти необычайно важные вопросы и посвящен данный эксперимент.

Цель эксперимента. Разобраться, как можно изменить естественное последовательное выполнение инструкций в программе с помощью переходов.

Для проведения экспериментов мы используем учебный компьютер E97, поскольку из всех рассмотренных в *разд. 8.6* моделей он наиболее близок к тому, как работает наиболее популярный у нас IBM PC.

Немного теории. Как вы, конечно, помните, после завершения в АЛУ очередной операции производится анализ полученного ответа *(см. разд. 3.1.2)*.

Примечание

Большинство процессоров производит указанный анализ после любой инструкции, а не только после арифметических действий или специализированной инструкции сравнения. Однако существуют исключения из этого правила. Например, в известном процессоре Z80 операции пересылки информации были реализованы без подобного анализа.

Для запоминания результатов анализа в процессоре имеются специальные бинарные управляющие признаки, входящие в состав единого регистра. Обычно в ЭВМ существует довольно большое количество управляющих битов; мы рассмотрим лишь два из них, а именно те, которые позволяют произвести сравнение двух чисел по полным математическим правилам.

Хотя рассматриваемые признаки имеются практически в каждой машине, их обозначения могут быть довольно разнообразными. Мы воспользуемся теми, что были приняты в ЭВМ семейства PDP [54], поскольку архитектура

этих компьютеров отличалась необыкновенной логичностью и прозрачностью. Отметим, что в модели E97 принята такая же система обозначений по той же самой причине.

Итак, в машинах с архитектурой PDP при сравнении числовых данных использовались два управляющих признака, влияющие на условные переходы — N и Z. Первый отображал факт отрицательности результата (от слова Negative — отрицательный). Фактически он дублировал содержимое знакового разряда, который для отрицательных чисел всегда установлен в единицу и равен нулю в противном случае. Второй отвечал за равенство или неравенство нулю (от слова Zero, т. е. ноль). Его значение устанавливалось в результате аппаратного анализа всех двоичных разрядов результата. Фактически признак Z — это закодированный двоичным образом ответ на вопрос, является результат нулевым: если ответ "да, является", то признак установлен в единицу, а если "нет", то, наоборот, сброшен в ноль.

Нетрудно сообразить, что при изолированном анализе двух бинарных признаков мы можем закодировать всего 4 возможных соотношения между двумя данными. В то же время, в математике их определено шесть! Выход состоит в совместном рассмотрении значений битов N и Z, как это показано в табл. 8.10.

Таблица 8.10. *Значения управляющих битов для условных переходов*

N	Логическая операция	Z	Действие	MOD E97
*	–	*	Безусловный переход	1
0	–	*	Переход при $R \geq 0$	2
1	–	*	Переход при $R < 0$	3
*	–	0	Переход при $R \neq 0$	4
*	–	1	Переход при $R = 0$	5
1	ИЛИ	1	Переход при $R \leq 0$	6
0	И	0	Переход при $R > 0$	7

Знак * показывает, что в данной ситуации значение управляющего бита может быть любым.

Из приведенной таблицы отчетливо видны 4 условия, которые реализуются в результате анализа изолированных признаков, и 2 последних, требующих их совместного рассмотрения. Очевидно, что при безусловном переходе состояние управляющих битов никакой роли не играет.

Отметим, что в табл. 8.10 предусмотрительно добавлен еще один, последний столбик, который нам пригодится позднее. Он содержит значения первой

цифры модификатора для компьютера E97, которые скоро нам потребуются для кодирования переходов в программе для эксперимента.

Таким образом, мы подробно выяснили, как происходит установка управляющих битов в зависимости от результата операции. Остается рассмотреть, каково дальнейшее использование этих битов инструкциями условных переходов.

Каждый условный переход предназначен для проверки состояния одного признака или определенной комбинации обоих. Алгоритм выполнения любого условного перехода следующий: если состояние признака (или признаков) соответствует "благоприятному" для перехода, то он производится. В противном случае условный переход игнорируется и обычным образом происходит выполнение следующей инструкции программы. Те читатели, кто знаком с программированием на языках высокого уровня, без труда узнали логику оператора `IF/THEN`. Только вот `ELSE` на уровне машинных команд отсутствует и его приходится реализовывать отдельно.

А сейчас перейдем непосредственно к эксперименту. Рассмотрим решение следующей задачи: пусть в регистрах `R1` и `R2` хранятся произвольные целые числа и требуется большее из них поместить в регистр `R0`. В табл. 8.11 приведено одно из возможных решений этой задачи.

Таблица 8.11. *Пример программы, реализующей безусловный переход*

Адрес	Код	Операция	Комментарий
0000	0412	Сравнить `R2` с `R1`	Сравнить числа
0002	3C0D	Если результат < 0,	Переход при отрицательном
0004	000C	то к 000C	результате к записи `R1`
0006	0120	`R2 ==> R0`	Запомнить `R2`
0008	1C0D	Безусловный переход	Обход второй ветви
000A	000E	к 000E	
000C	0110	`R1 ==> R0`	Запомнить `R1`
000E	0F00	Останов	

Рассмотрим подробнее первые две операции, которые представляют наибольший интерес с точки зрения реализуемого сейчас опыта.

Программа начинается командой сравнения двух регистров `R1` и `R2`, в которых лежат исходные данные. Это специализированная инструкция процессора для подготовки значений управляющих признаков, за которой всегда следует команда условного перехода. Суть сравнения состоит в установке битов N и Z в соответствии с результатом сравнения, причем последний

больше никуда не записывается и после операции теряется, так что обе сравниваемые величины остаются без изменения.

Далее, как и следует из общей теории, стоит инструкция условного перехода. Заглянув в табл. 8.10, по первой цифре (модификатору MOD) мы узнаем, что это переход в случае, если результат сравнения является отрицательным. Иными словами, переход по указанному в команде адресу 000C в нашей программе будет происходить при R2 < R1 и игнорироваться — в противном случае. Читатели могут самостоятельно убедиться в том, что такая логика обеспечивает занесение большего из двух сопоставляемых значений в R0.

Нам остается лишь реализовать полученную программу на виртуальной ЭВМ E97. Разрешите предположить, что те читатели, которые захотят это проделать, уже повторили эксперимент, описанный в *разд. 8.6.2*, что позволит не описывать повторно принципы работы с моделью E97. Напомню только, что ввод программы осуществляется по директиве монитора S (можно ввести уже набранный файл; для данного эксперимента он имеет имя E8_4). Далее обязательно надо задать значения регистров R1 и R2, которые служат исходными данными в нашей задаче. Для реализации этой операции необходимо выполнить директивы XR1 и XR2, которые работают по тем же самым принципам, что и директива ввода в ОЗУ. После этого останется набрать директиву G, запускающую программу с начального (нулевого) адреса.

Автор убедительно рекомендует выполнить программу не менее двух раз, причем так, чтобы в одном случае R1 было больше R2, а в другом — наоборот (очевидно, что случай равенства в данной задаче особого интереса не представляет). Только после этого можно быть уверенным, что программа работает правильно.

Для тех, кто сумел успешно проделать все описанное выше, предлагается последний достаточно тонкий, но очень полезный в теоретическом отношении эксперимент: пошаговое выполнение сравнения и условного перехода с целью анализа максимально возможного количества деталей. Протокол выполнения эксперимента для R1 = 1 и R2 = 2 приведен на рис. 8.8.

Как и раньше, для удобства нахождения нужной информации на копию экрана нанесена вспомогательная разметка, облегчающая читателю расшифровку полученных результатов. Итак, в начальной части протокола мы видим ввод указанных выше значений в регистры E97. Прямоугольная рамка после этих операций выделяет то место "распечатки", где видно, что операция прошла успешно, и регистры действительно получили требуемые нам значения. Далее следует принципиальный шаг — задание начального значения программного счетчика PC (при полном выполнении программы по директиве G это делать не требовалось). Поскольку мы собираемся "трассировать" программу с самого начала, введем адрес 0. Последнее наше действие состоит в том, чтобы директивой T2 попросить учебный компьютер E97 исполнить 2 инструкции программы. В ответ на наши действия монитор

выведет на экран результаты исполнения каждой инструкции. Советуем особое внимание обратить на значения управляющих битов N и Z, установленных инструкцией сравнения, и содержимое счетчика команд PC после перехода: очевидно, что переход при данных значениях регистров не состоялся, и поэтому будет выполняться естественным образом следующая команда программы по адресу 6.

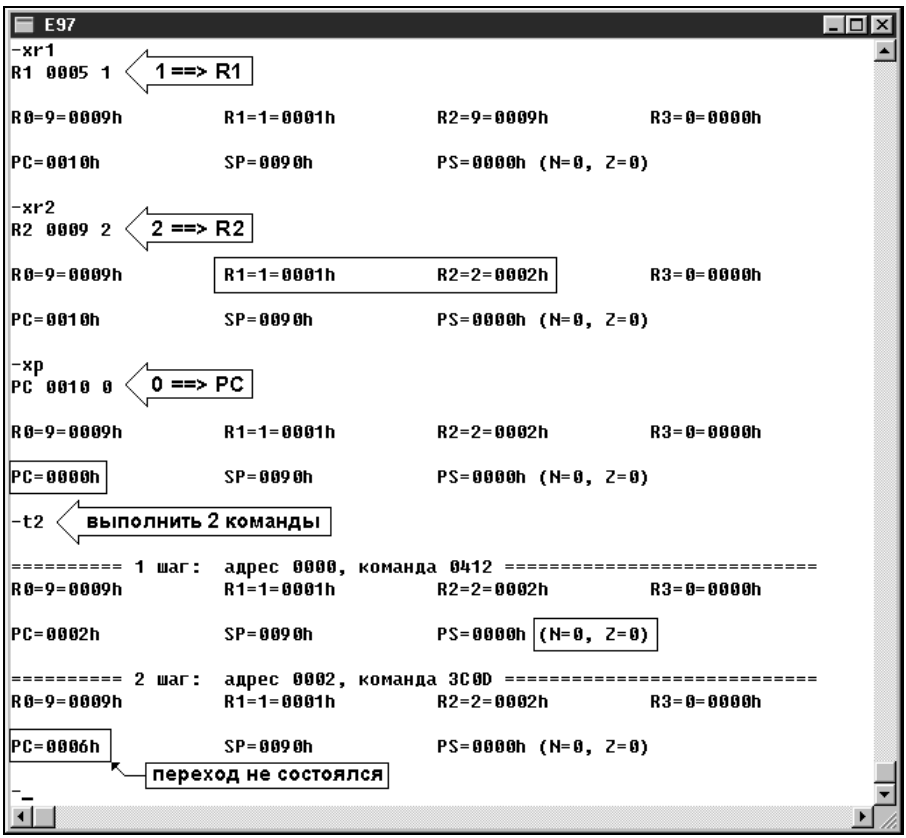


Рис. 8.8. Пошаговый анализ программы для случая, когда переход не состоялся

Совершенно аналогично можно проделать трассировку для противоположного случая, когда `R1 > R2` и переход произойдет. Предлагаем читателям проделать эту часть опыта самостоятельно, используя в качестве опоры рис. 8.9.

Выводы. Таким образом, мы видели, как реализуется условный переход в учебном компьютере E97. По такой же логике осуществляет условные переходы процессор Intel (если читатели захотят проверить данное утверждение, можно обратиться к одной из многочисленных книг по IBM PC; автор советует выбрать книгу [62]).

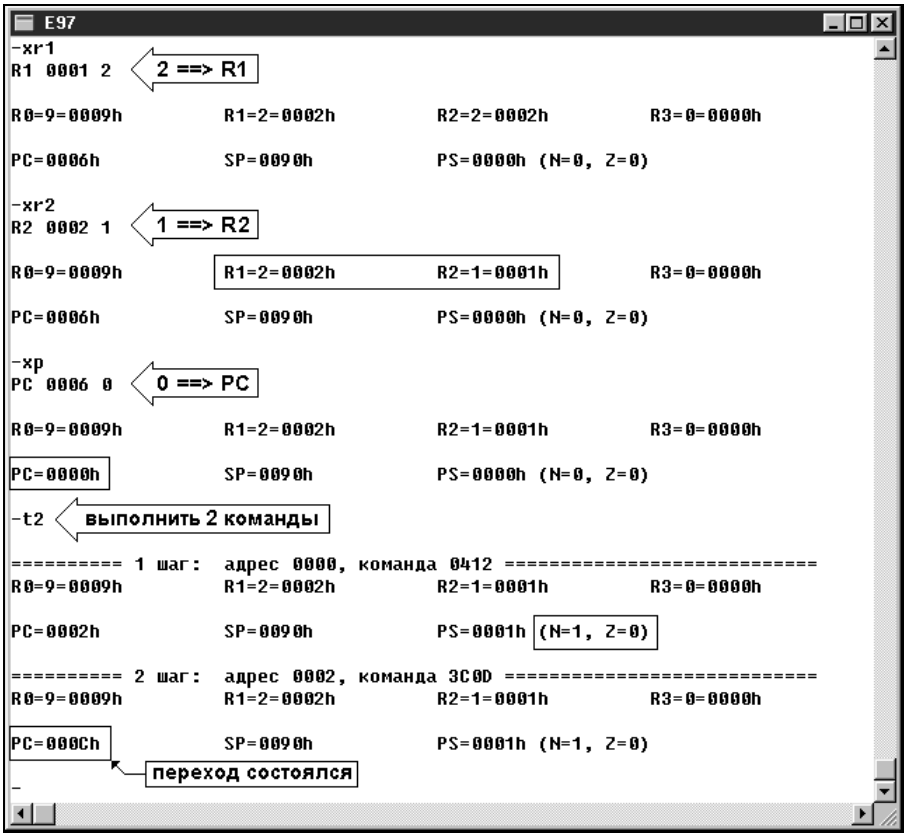


Рис. 8.9. Пошаговый анализ программы для альтернативного случая, когда переход состоялся

При реализации ветвления в программе ЭВМ "руководствуется" следующим алгоритмом. Сначала выполняется некоторая команда, результат которой должен определить, "быть или не быть" условному переходу. Чаще всего, это операция сравнения, но возможны и другие варианты (например, на практике может использоваться вычитание единицы из счетчика повторений с целью последующего анализа полученной разности на нуль). Во всех случаях по результату той или иной операции устанавливаются специальные управляющие биты. Их анализ и производится инструкцией условного перехода. При совпадении значения признака (или комбинации признаков) с требуемым для данной конкретной инструкции процессор изменяет содержимое программного счетчика и тем самым нарушает естественный порядок исполнения программы. Именно такой случай и называется переходом. Если же, наоборот, условие не было удовлетворено, то условный переход игнорируется и обычным образом выполняется следующая по программе инструкция.

Подчеркнем, что на основе условного перехода (или комбинации нескольких переходов, в том числе, возможно, включая безусловный, как это было в программе данного эксперимента) реализуются все алгоритмические структуры. Сюда относятся полная и неполная форма развилки, выбор (имеются в виду конструкции типа `CASE` в языке Паскаль) и все виды циклов. По мнению автора, любой грамотный человек, который хочет понимать, каким образом современный компьютер принимает те или иные решения, обязан знать наиболее общие фундаментальные принципы работы операции условного перехода, изложенные выше.

8.6.5. Как работает язык высокого уровня

Выбранные в качестве иллюстраций к данной лекции эксперименты, разумеется, не случайны. О том, что первые три из них образуют единую последовательность, направленную на изучение трех-, двух- и одноадресных ЭВМ, было очевидно с самого начала. Другая логическая последовательность состояла в сопоставлении первых трех экспериментов с четвертым, что позволило понять, как компьютер реализует линейный процесс вычислений, а также остальные алгоритмические структуры (развилку и цикл). Последний эксперимент завершит еще одну содержательную линию, которая, возможно, не так очевидна: работа с машинным кодом в *разд. 8.6.1 и 8.6.2*, Ассемблер в *разд. 8.6.3* и, наконец, язык высокого уровня в данном эксперименте.

Цель эксперимента. Посмотреть, что представляет собой язык высокого уровня и как текст программы на нем транслируется в непосредственно исполняемый машинный код.

Для решения поставленной задачи воспользуемся учебным компилятором Паскаля КомПас [27, 28], который работает на виртуальном компьютере E97. Его реализация находится на прилагаемом к книге CD-диске в папке с соответствующим названием.

Примечание

Поскольку КомПас — это достаточно сложное Windows-приложение, его пришлось для компактности поместить на CD-диск в виде ZIP-архива (это единственное исключение: все остальные файлы не сжимались и немедленно готовы к использованию). Автор искренне надеется, что читатель, который заинтересуется работой языка высокого уровня, сумеет развернуть стандартный архив без пошаговых инструкций. Это единственное, что требуется сделать, поскольку КомПас не требует специальной инсталляции и готов к работе сразу же после извлечения из архива.

На этот раз интерфейс программного обеспечения будет более дружественным, что сделает реализацию эксперимента гораздо более простой. В книге будет описана трансляция всего одного оператора из единственной реализо-

ванной программы. Те читатели, которые заинтересуются работой компилятора, могут воспользоваться электронной документацией, входящей в комплект программы (детальный файл помощи и около 50 встроенных разобранных примеров), а также очень подробной публикацией [28], которая напечатана достаточно большим тиражом в известной педагогической газете "Информатика".

Начнем с того, что еще раз проследим общую содержательную линию, о которой говорилось во введении к данному эксперименту. Напомним, что до сих пор единственным языком, который способен понимать процессор непосредственно, является двоичный машинный код. Однако работа с длинными последовательностями, состоящими из нулей и единиц, для человека крайне неудобна. Поэтому уже на первых огромных неуклюжих ЭВМ стали появляться всевозможные средства автоматизации процесса кодирования программ. Вместо громоздкой двоичной системы почти сразу же стала использоваться более компактная кратная ей восьмеричная, что немедленно сократило количество символов в машинной команде втрое (как известно, каждая восьмеричная цифра заменяет собой три двоичных: $8 = 2^3$). Позднее, когда появилась байтовая организация памяти, восьмеричная система "вступила с ней в конфликт", поскольку в байте оказалось нецелое количество восьмеричных цифр. Зато шестнадцатеричная система счисления идеально гармонировала с байтом и постепенно полностью вытеснила восьмеричную.

Даже такое, казалось бы, безобидное усовершенствование, как ввод шестнадцатеричных цифр, уже требует наличия специальной системной программы, которая осуществляет перевод во внутреннее двоичное представление. Если программу ввода дополнить некоторыми минимальными сервисными функциями (запуск программы с заданного адреса, просмотр содержимого памяти и некоторые другие), то мы получим не что иное, как программу-монитор, с которой читатели уже знакомы не только в теории, но и на практике. Как мы видели, работая в мониторе, можно набирать и отлаживать несложные программы в командах процессора. Но работа эта, как сейчас модно говорить, "не для всех". В самом деле, реализация программы на уровне монитора требует большой концентрации внимания и, что самое главное, полученные программы очень трудно усовершенствовать: достаточно вставить всего одну команду и программа сдвигается, так что входящие в нее адреса приходится аккуратно заменять на новые. А это уже значительно труднее, чем просто без опечаток ввести с листка бумаги столбик цифр!

Конечно, выход был найден и из этой ситуации. Поскольку наиболее трудоемкой оказалась именно "привязка" программы к конкретным адресам памяти, то и этот процесс тоже поручили машине. Появился язык Ассемблер, с образцом которого мы уже знакомы по работе с учебной ЭВМ Малютка *(см. разд. 8.6.3)*. Важнейшим достоинством Ассемблера по сравнению с программой в машинных кодах является отказ от "ручного" планирования и использования адресов ОЗУ и их замена символьными идентификаторами.

Возможно, это не всем очевидно, но главное преимущество идентификатора состоит вовсе не в том, что его легче запомнить, чем числовой адрес. Существенно важнее то, что при изменении программы идентификатор, в отличие от соответствующего ему адреса памяти, сохраняется, а это значит, что программу несоизмеримо легче переделать.

Примечание

Для проверки этого утверждения можете попробовать вставить еще одну переменную между `x` и `b` в программу для MMIX или E97 (подход машинных кодов) и для Малютки с ее встроенным Ассемблером.

Говоря об Ассемблере, стоит подчеркнуть, что любая его команда однозначно соответствует команде процессора. Так что вы по-прежнему программируете в командах машины, только делаете это значительно более комфортабельно. Следующий принципиально новый шаг состоит в том, что наиболее часто встречающиеся группы команд (например, вывод на экран) оформляют в виде стандартных *макроопределений* или, как часто говорят для краткости, *макросов*. В программе вместо данной группы команд достаточно написать ее название, а расширенная система, которая называется *макроассемблер*, подставит вместо этой строчки необходимую совокупность команд. Таким образом, строка программы на макроассемблере теперь может преобразовываться не только в одну машинную инструкцию, но и в их определенную последовательность. В качестве последнего шага остается лишь определить наиболее часто встречающиеся при программировании действия и их эквивалентную замену в машинных кодах, и мы получим определенный язык программирования. Его называют *языком высокого уровня*, чтобы подчеркнуть, что он не содержит в явном виде машинных команд, как это было в Ассемблере. Очевидно, что сам по себе способ написания программы на языке высокого уровня не зависит от того, на какой машине он реализуется.

Отметим, что существует два распространенных способа трансляции (перевода) программы, написанной на языке высокого уровня: компиляция (полный перевод) и интерпретация (перевод по частям с немедленным исполнением). Мы для простоты ограничимся только первым из них.

Выяснив основную идею, попробуем посмотреть, как это выглядит на практике. К сожалению, разобраться, как работает реальный компилятор нам вряд ли удастся из-за сложности последнего. Поэтому вновь воспользуемся специальным учебным ПО, которое продемонстрирует нам процесс компиляции программы с языка Паскаль в код процессора E97.

Подготовим несложную программу на языке Паскаль. Например, следующую:

```
PROGRAM proba;
CONST x=2;
VAR   y:INTEGER;
```

```
BEGIN y:=2*x;
      WRITELN(y);
END.
```

Наберем ее в левом поле программы КомПас (или просто загрузим из файла E8_5) и щелкнем по кнопке **Compile**. В результате получим на экране картину, аналогичную приведенной на рис. 8.10.

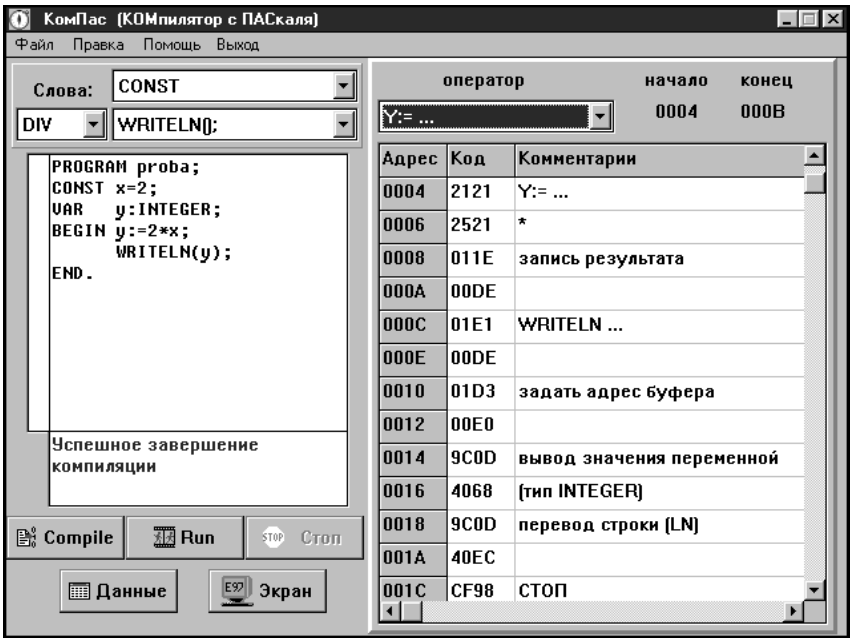


Рис. 8.10. Изучение компиляции на базе учебной программы КомПас

Разберем подробно результаты трансляции всего одного оператора, а именно `y := 2 * x`. Для этого выберем его в списке операторов (на рис. 8.10 этот список легко найти в верхней части окна справа над таблицей с кодом программы: на нем находится фокус ввода). Затем аккуратно выпишем результаты трансляции этого оператора и разберем их (табл. 8.12).

Таблица 8.12. *Результаты трансляции программы в исполняемый код*

Адрес	Код	Операция	Комментарий
0004	2121	`2 ==> R1`	Коэффициент 2
0006	2521	`R1 * 2 ==> R1`	Умножить на const `x`
0008	011E	`R1 ==> (DE)`	Записать результат в ОЗУ
000A	00DE		По адресу переменной `y`

Мне кажется, вы будете приятно удивлены тем, что ничего сложного для себя не обнаружите (в предыдущих опытах мы разбирали программы посложнее!) Сначала в R1 извлекается первый операнд, а затем он умножается на константу x. Наконец, заключительная, третья команда, записывает получившееся число в ячейку ОЗУ, которая соответствует переменной y. Последний факт легко проверить, вызвав на экран таблицу распределения памяти с помощью кнопки **Данные**.

При разборе оператора WRITELN следует иметь в виду, что КомПас использует стандартные подпрограммы из ПЗУ. Детали здесь мы обсуждать не будем, поскольку подробная информация по ним приведена в файле помощи.

> **Выводы.** Исходной информацией для компилятора является записанный по определенным правилам текст. В результате его анализа, если он не содержал ошибок, компилятор автоматически составляет эквивалентную программу в командах процессора, которая может быть исполнена или сохранена в виде стандартного исполняемого файла. После завершения трансляции исходный текст на языке высокого уровня становится ненужным, а результирующая программа способна работать самостоятельно.

Вот и завершился наш самый последний эксперимент. Автор очень надеется, что в целом читателям были интересны описанные в книге эксперименты и по крайней мере от повторения некоторых они получили удовольствие. А если в ходе проведенных опытов у читателей появилось стремление проверять теоретические знания, придумывая свои собственные эксперименты на компьютере, то значит, что достаточно большое время, которое ушло у автора на разработку, отладку и написание "экспериментальной" части книги, тем более было затрачено не зря.

ПРИЛОЖЕНИЕ

Содержание CD-диска

Внимание!

Прилагаемый CD-диск в приложениях к *лекции 7* содержит файл EICAR.com, который служит для проверки антивирусных программ. Он **не является вирусом**, хотя на него и реагирует антивирусное ПО!!! (Если есть сомнения — внимательно прочтите подробности в *разд. 7.10.6* и посетите приведенные в конце его интернет-ссылки.)

Прилагаемый CD-диск предназначается для проведения описанных после лекций любопытных экспериментов. Сразу оговорим, что это ***приложение*** к книге и ее самостоятельное использование без текста книги в большинстве случаев проблематично.

Помещенные на диске файлы главным образом являются **текстами программ**, листинги которых приведены в тексте, и **результатами их трансляции** в исполняемые файлы. Использование этих файлов позволяет при желании сэкономить время на наборе текста программ или отладке (все программы на диске тщательно проверены и заработают сразу же).

Примечание

С точки зрения освоения языка программирования, разумеется, лучше написать, набрать, отладить и запустить программу самостоятельно. Но изучение программирования не является целью данной книги, поэтому "сокращенный" вариант запуска "готовой" программы, которая предварительно была внимательно разобрана по тексту книги, вполне допустим.

Другой вариант использования CD-диска — это проведение экспериментов в случае, когда читатели по тем или иным причинам не знакомы с программированием, но тем не менее описанные занимательные компьютерные опыты хотели бы повторить. Тогда можно, "поверив автору на слово", просто запустить уже откомпилированный исполняемый файл (учтите только, что в последнем случае вы теряете возможность внести какие-либо собственные изменения в проведение эксперимента!) В частности, один из любо-

пытных экспериментов *(см. разд. 6.7.4)* реализован в среде Delphi. Очевидно, что далеко не у всех на компьютере установлена эта современная система, но наличие на CD-диске исполняемого файла позволит своими глазами увидеть описанные в книге результаты каждому читателю.

Особая ситуация с приложениями к последней лекции. В предназначенном для нее каталоге находится авторское программное обеспечение, реализующее некоторые учебные модели компьютера. Таким образом, в отличие от всех предыдущих лекций, провести эксперименты без диска с ПО здесь в принципе нельзя. Отметим, что возможности прилагаемых учебных программ гораздо шире, чем это использовано в описанных в книге экспериментах. Сказанное в особенности относится к демонстрационному компилятору Паскаля КомПас *(см. разд. 8.6.5)*, который является оригинальным и достаточно уникальным программным продуктом.

Находящиеся на CD-диске файлы сгруппированы по лекциям, причем именем каталога служит номер лекции. Для удобства работы с листингами, файлы имеют названия, совпадающие с нумерацией листингов в тексте книги. Например, листинг 2.1, который является первым в *лекции 2*, находится в файле LIST_2_1 и т. д.

Примечание

Следует учитывать, что тексты программ на Паскале и некоторые другие находящиеся на CD-диске файлы по необходимости хранят русские символы в кодировке MS-DOS. Чтобы прочитать их, можно воспользоваться программой Far или любым другим ПО, позволяющим отобразить содержимое текстового файла в кодировке DOS.

В каждом из восьми каталогов имеется собственный текстовый файл ReadMe, содержащий описание назначения каждого файла и некоторые пояснения.

Все файлы готовы к немедленному использованию (как непосредственно с CD-диска, так и после стандартного копирования с жесткого диска). Исключение составляет уже упоминавшийся компилятор КомПас. Это довольно большое Windows-приложение, занимающее около 1 Мбайт, поэтому его пришлось поместить в архив. Перед запуском архивный ZIP-файл достаточно стандартным образом распаковать на жесткий диск вашего компьютера — никаких других дополнительных манипуляций не требуется.

Список литературы

1. Абель П. Язык Ассемблера для IBM PC и программирования. — М.: Высшая школа, 1992. — 447 с.
2. Абрамов П. Имитатор процессора // Информатика и образование. 1986, № 2, с. 74—77.
3. Андреева Е., Фалина И. Информатика: Системы счисления и компьютерная арифметика. — М.: Лаборатория Базовых Знаний, 1999. — 256 с.
4. Антипов И. Н. Учебная модель ЭВМ // Математика в школе. 1977, № 2, с. 55—59.
5. Пакет программной поддержки школьного учебника информатики для КУВТ "УКНЦ" / Бирих Р. В., Бирих В. М., Еремин Е. А., Еремина Г. П., Люшнин А. В., Чернатынский В. И., Шабуров В. П. // Тез. докл. Респ. конф. Пермь, 1994, с. 114—115.
6. Борисов В., Партин А. Основы цифровой техники // Радио, 1985, № 7, с. 50—51.
7. Бродин В. Б., Шагурин И. И. Микропроцессор i486. Архитектура, программирование, интерфейс. — М.: Диалог-МИФИ, 1993. — 240 с.
8. Брукшир Дж., Глен. Введение в компьютерные науки. Общий обзор. — М.: Издательский дом "Вильямс", 2001. — 688 с.
9. Букчин Л. В., Безрукий Ю. Л. Дисковая подсистема IBM-совместимых персональных компьютеров. — М.: Бином, 1993. — 284 с.
10. Васильев Б. М., Частиков А. П. Микропроцессоры (история, применение, технология) // Информатика и образование. 1993, № 5, с. 71—83.
11. Основы информатики и вычислительной техники: Проб. учеб. пособие для сред. учеб. заведений / А. Г. Гейн, В. Г. Житомирский, Е. В. Линецкий и др. — Свердловск: Изд-во Урал. ун-та, 1989. — 272 с.
12. Гинзбург А., Милчев М., Солоницын Ю. Периферийные устройства. — СПб.: Питер, 2001. — 448 с.
13. Гринзоу Л. Философия программирования для Windows 95/NT. — СПб.: Символ-Плюс, 1997. — 640 с.
14. Гук М. Ю. Процессоры Intel: от 8086 до Pentium II. — СПб.: Питер, 1997. — 224 с.
15. Гук М. Ю. Аппаратные средства IBM PC. Энциклопедия. — СПб.: Питер, 2000. — 816 с.
16. Гук М. Ю. Дисковая подсистема ПК. — СПб.: Питер, 2001. — 336 с.
17. Гутер Р. С., Полунов Ю. Л. От абака до компьютера. — М.: Знание, 1975. — 192 с.
18. Джордейн Р. Справочник программиста персональных компьютеров типа IBM PC, XT, AT. — М: Финансы и статистика, 1992. — 554 с.
19. Дроздов Д. Е., Комарницкий В. А., Пятибратов А. П. Электронные вычислительные машины Единой системы. — М.: Машиностроение, 1981. — 648 с.
20. Дунаев С. UNIX SYSTEM V. RELEASE 4.2. — М.: Диалог-МИФИ, 1995. — 287 с.
21. Дьяконов В. П. Мой Pentium. — М.: ООО Издательство АСТ-ЛТД, 1998. — 536 с.
22. Еремин Е. А. Программа для изучения принципов работы процессора персональной ЭВМ // Тез. докл. Всесоюз. конф. Омск, 1987, с. 123—125.
23. Еремин Е. А., Пономарева Л. В. Методические рекомендации по изучению основных принципов работы персонального компьютера. — Пермь: ПГПИ, 1989. — 54 с.

24. Еремин Е. А. Компилятор? Это очень просто // Компьютер УКНЦ. М.: Компьютика, 1995, № 3, с. 25—33.

25. Еремин Е. А. Как работает современный компьютер. — Пермь: изд-во ПРИПИТ, 1997. — 176 с.

26. Еремин Е. А. Windows'95 // Информатика и образование. 1997, № 4, с. 37—41; № 5, с. 88—96.

Статья доступна в Интернет по адресу: http://e-eremin.nm.ru/public/info/win95/p0.html.

27. Еремин Е. А. Компилятор? Это довольно просто! — Пермь: Изд-во ПРИПИТ, 1998. — 120 с.

28. Еремин Е. А. Компилятор? Это довольно просто! // Информатика. 2001, № 40, с. 7—17; № 43, с. 7—14; № 45, с. 21—29; № 46, с. 19—25; № 47, с. 8—10.

29. Еремин Е. А. Основы вычислительной техники // Информатика. 2002, № 33, с. 2—9; № 35, с. 2—6; № 37, с. 3—9; № 39, с. 3—9; № 41, с. 3—9; № 43, с. 4—10; № 45, с. 3—10, № 47, с. 3—9.

30. Еремин Е. А. MMIX — учебный RISC-процессор нового тысячелетия от Дональда Кнута. // Информатика. 2002, № 40, с. 18—27.

Статья доступна в Интернет по адресу: http://emc.km.ru/mmix/mmixart.html.

31. Изучение основ информатики и вычислительной техники: Метод. пособие для учителей и преподавателей сред. учеб. заведений. В 2 ч. Ч. 2 / А. П. Ершов, В. М. Монахов, М. В. Витиньш и др.; Под ред. А. П. Ершова, В. М. Монахова. — М.: Просвещение, 1986. — 207 с.

32. Основы информатики и вычислительной техники: Проб. учеб. пособие для сред. учеб. заведений. В 2 ч. Ч. 2 / А. П. Ершов, В. М. Монахов, А. А. Кузнецов и др.; Под ред. А. П. Ершова, В. М. Монахова. — М.: Просвещение, 1986. — 143 с.

33. Информатика в понятиях и терминах / Под ред. В. А. Извозчикова. — М.: Просвещение, 1991. — 208 с.

34. Информатика / Под ред. Н. В. Макаровой. — М.: Финансы и статистика, 2001. — 768 с.

35. Информатика: Энциклопедический словарь для начинающих / Сост. Д. А. Поспелов. — М.: Педагогика-Пресс, 1994. — 352 с.

36. Гибкие диски // Информатика. 2002, № 5, с. 16—17.

37. Музыкальный компакт-диск // Информатика. 2002, № 6, с. 16—17.

38. Привод CD-ROM // Информатика. 2002, № 7, с. 16—17.

39. Winchester // Информатика. 2002, № 8, с. 16—17.

40. DVD-диски // Информатика. 2002, № 14, с. 16—17.

41. Жидкокристаллические дисплеи // Информатика. 2002, № 16, с. 16—17.

42. Плазменные дисплеи // Информатика. 2002, № 19, с. 16—17.

43. В поисках пикселя или Типы электронно-лучевых трубок // Информатика. 2002, № 47, с. 16—17.

44. Мыши // Информатика. 2003, № 4, с. 16—17.

45. Иртегов Д. В. Введение в операционные системы. — СПб.: БХВ-Петербург, 2002. — 624 с.

46. Основы информатики и вычислительной техники: Проб. учеб. пособие для 10—11 кл. сред. шк. / В. А. Каймин, А. Г. Щеголев, Е. А. Ерохина и др. — М.: Просвещение, 1989. — 272 с.

47. Карцев М. А. Арифметика цифровых машин. — М.: Наука, 1969. — 576 с.

48. Керниган Б. В., Пайк Р. UNIX — универсальная среда программирования. — М.: Финансы и статистика, 1992. — 303 с.

49. Кнут Д. Искусство программирования на ЭВМ. Том 1. — М.: Мир, 1976. — 736 с.

50. Кушниренко А. Г., Лебедев Г. В., Сворень Р. А. Основы информатики и вычислительной техники: Проб. учеб. для сред. учеб. заведений. — М.: Просвещение, 1990. — 224 с.

51. Информатика. 7—9 кл.: Учеб. для общеобразоват. учеб. заведений / А. Г. Кушниренко, Г. В. Лебедев, Я. Н. Зайдельман. — М.: Дрофа, 2000. — 336 с.

52. Информационная культура: Кодирование информации. Информационные модели. / А. Г. Кушниренко, А. Г. Леонов, М. Г. Эпиктетов и др. — М.: Дрофа, 2000. — 208 с.

53. Кушниренко А. Г. Оставьте информатику в школе, а какой ей быть — отрегулирует сама жизнь // Информатика. 2001, № 19, с. 2—3.

54. Лин В. PDP-11 и VAX-11. Архитектура ЭВМ и программирование на языке ассемблера. — М.: Радио и связь, 1989. — 320 с.

55. Малая универсальная вычислительная машина "Наири-2". Техническое описание. Ч. 1. Краткое описание и система команд. 1968. — 121 с.

56. Математический энциклопедический словарь. / Гл. ред. Ю. В. Прохоров. — М.: Сов. энциклопедия, 1988. — 847 с.

57. Мнеян М. Г. Физические принципы работы ЭВМ. — М.: Просвещение, 1987. — 192 с.

58. Могилев А. В., Пак Н. И., Хеннер Е. К. Информатика: учеб. пособие для студ. пед. вузов. — М.: Академия, 1999. — 816 с.

59. Могилев А. В., Пак Н. И., Хеннер Е. К. Практикум по информатике: учеб. пособие для студ. высш. учеб. заведений. — М.: Академия, 2001. — 608 с.

60. Нестеренко А. В. ЭВМ и профессия программиста. — М.: Просвещение, 1990. — 160 с.

61. Нортон П. Персональный компьютер фирмы IBM и операционная система MS-DOS. — М.: Радио и связь, 1992. — 416 с.

62. Нортон П., Соухэ Д. Язык ассемблера для IBM PC. — М.: Компьютер, 1992. — 352 с.

63. Сетевые операционные системы / В. Г. Олифер, Н. А. Олифер. — СПб.: Питер, 2002. — 544 с.

64. Основы информатики и вычислительной техники в базовой школе: Пособие для учителя / Под ред. И. Г. Семакина. — Пермь, 1995. — 282 с.

65. Островский С. Л. Компьютерные вирусы // Информатика. 2001, № 32, с. 1—29.

66. Отставнов М. Свободное программное обеспечение // Информатика. 2002: № 37, с. 10—15; № 39, с. 10—11, 22—25; № 41, с. 10—13; № 43, с. 11—15; № 45, с. 11—14; № 47, с. 10—13; 2003: № 1, с. 9—14; № 3, с. 7—11; № 5, с. 8—12; № 7, с. 7—11; № 9, с. 11—15; № 11, с. 21—24; № 15, с. 19—23; № 17, с. 8—12.

67. Петцольд Ч. Код. — М.: Русская Редакция, 2001. — 512 с.

68. Пресс Б., Пресс М. Ремонт и модернизация ПК. Библия пользователя. — М.: Издательский дом "Вильямс", 2000. — 1120 с.

69. Решетников В. Н., Сотников А. Н. Информатика — что это? — М.: Радио и связь, 1989. — 112 с.

70. Информатика. Учебник по базовому курсу. / Семакин И. Г., Залогова Л. А., Русаков С. В. и др. — М.: ООО "Издательство лаборатория базовых знаний", 1998. — 464 с.

71. Сенокосов А. И., Гейн А. Г. Информатика: Учеб. для 8—9 кл. шк. углуб. изуч. информатики. — М.: Просвещение, 1995. — 225 с.

72. Симаненков Д. Из аналога в цифру и обратно: немного теории // Компьютерра. 1998, № 30—31, с. 22—27.

Статья доступна в Интернет по адресу: http://www.computerra.ru/offline/1998/258/1481/.

73. Смит Б. Э., Джонсон М. Т. Архитектура и программирование микропроцессора INTEL 80386. — М.: Конкорд, 1992. — 334 с.

74. Справочник по полупроводниковым диодам, транзисторам и интегральным схемам / Под ред. Н. Н. Горюнова. — М.: Энергия, 1977. — 744 с.

75. Стахнов А. А. Linux. — СПб.: БХВ-Петербург, 2002. — 912 с.

76. Угринович Н. Информатика и информационные технологии. Учебное пособие для общеобразовательных учреждений. — М.: БИНОМ, 2001. — 464 с.

77. Уэйт М., Ангермейер. Дж. Операционная система CP/M. — М.: Радио и связь, 1986. — 312 с.

78. Частиков А. П. История компьютера. — М.: Информатика и образование, 1996. — 128 с.

79. Частиков А. П. Архитекторы компьютерного мира. — СПб.: БХВ-Петербург, 2002. — 384 с.

80. От тюрьмы да от сумы // Чип. 2002, № 10, с. 13.

81. Шафрин Ю. А. Информационные технологии: Ч. 1. — М.: Лаборатория Базовых Знаний, 2000. — 320 с.

82. Knuth D. E. MMIXware: A RISC Computer for the Third Millennium (Lecture Notes in Computer Science, no. 1750). Heidelberg: Springer-Verlag, 1999, 550 pp.

Ссылки на интернет-страницы

- И-1. Анатомичка: жесткие диски. Upgrade, 2002, № 11 (49). http://www.computery.ru/upgrade/numbers/2002/049/hard_2_49.htm
- И-2. Афанасьев К. Мыши становятся быстрее. http://kv.minsk.by/index1999193901.htm
- И-3. Вирусная энциклопедия. Eicar. http://www.viruslist.com/viruslist.html?id=3242
- И-4. Громов В. И., Васильев В. А. Энциклопедия компьютерной безопасности. http://lib.rin.ru/cgi-bin/index.pl?open=27&view=5580&showby=1&sortby=1&letter=&from=8
- И-5. Ерохин А. Принтеры. Краткий курс. — Компьютерра, 2002, № 4. http://www.computerra.ru/special/2002/4/20331/index.html
- И-6. "Лаборатория Касперского" о вирусе "Анна Курникова". http://www.rol.ru/news/it/news/01/02/13_028.htm
- И-7. Машина для инженерных расчетов МИР-2.
- http://www.icfcst.kiev.ua/MUSEUM/PHOTOS/MIR2_r.html
- И-8. Музыченко Е. Винчестер снаружи и изнутри. http://spider.nrcde.ru/music/articles/hardware/hdd_outsins.html
- И-9. Мул Ю., Поперечный В. Винчестер под микроскопом. http://rol.ru/news/it/helpdesk/nk/
- И-10. Мыши бывают разные. http://anews.ru/mouse.html

- И-11. Паршин М. М. Отладчик для E97.
  http://www.geocities.com/m_parshin/debuger
- И-12. Пашков Д. Жесткие диски изнутри.
  http://old.submarine.ru/win/598/hard2_1.html
- И-13. Примечания к программе NORTON ANTIVIRUS.
  http://a21.viptop.ru/soft/Nantvir.htm
- И-14. Программная реализация: "Малютка".
  http://inf.1september.ru/eremin/emc/malutka/download.html
- И-15. Проект rEd-MMI.
  http://emmi.4u.ru
- И-16. Ссылки.
  http://pdp11.org.ru/links.html
- И-17. Станкевич Н. Обзор манипуляторов. Часть 1 — мышиное царство.
  http://price.od.ua/freehtml/articles27.htm
- И-18. Учебные модели ЭВМ.
  http://emc.km.ru
- И-19. Черняк Л. Изобретатель мыши, но не только.
  http://www.computer-museum.ru/frgnhist/mouse.htm
- И-20. Active SMART — узнайте о состоянии вашего жесткого диска.
  http://www.ariolic.ru/activesmart
- И-21. Brain M. How Computer Mice Work.
  http://www.howstuffworks.com/mouse2.htm
- И-22. Cpu-Z версии 1.14a.
  http://www.fcenter.ru/articles.shtml?utilities/4426
- И-23. EICAR — Anti-Virus test file (на английском языке).
  http://www.eicar.org/anti_virus_test_file.htm
- И-24. Intel® Processor Frequency ID Utility.
  http://www.intel.com/support/processors/tools/frequencyid/freqid.htm
- И-25. MMIX 2009 — a RISC computer for the third millennium
  (официальная страница MMIX; на английском языке).
  http://www-cs-faculty.stanford.edu/~knuth/mmix.html
- И-26. PDP-11 за железным занавесом.
  http://pdp11.ru/cgi-bin/get_p.cgi?te=iron
- И-27. SMARTUDM
  http://www.sysinfolab.com/ru/smartudm.htm
- И-28. William T. Verts. An Essay on Endian Order (на английском языке).
  http://www.cs.umass.edu/~verts/cs32/endian.html
- И-29. WinHex: Hex Editor for Files, Disks & RAM.
  http://www.winhex.com/winhex/index-m.html

Предметный указатель

М

О

П

Р

С

Т

У

Ф

Ш

Я