Программно-аппаратная организация компьютера IBM PC

Prentice-Hall Publishing Comp.,1984
П.Нортон. Перевод с английского С.Е.Писарева

Предисловие переводчиков

Авторы перевода благодарны всем им за оказанную поддержку и внимание.

В дальнейшем перевод будет изменяться и пополняться с учетом материалов (отсутствующих в оригинале), отражающих

ОГЛАВЛЕНИЕ

Глава 1. ВВЕДЕНИЕ В СФЕРУ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

[Оглавление]

Книга, предлагаемая читателю, рассказывает о чудесах - о тех чудесах, которые позволяет творить Персональный Компьютер фирмы "IBM" (IBM/PC).

Появление персонального компьютера фирмы "IBM" знаменовало собой фактическое введение нового и очень высоко- го стандарта качества и производительности персональных компьютеров. Те, кто уже знал и понимал возможности персональных компьютеров, увидели в IBM/PC новое средство, превосходящее все то что существовало до сих пор. Те же, кто считал персональные компьютеры не более чем игрушками, начали осозновать действительную ценность этих компьютеров в качестве полезнейшего рабочего инструмента инженера или ученого.

Эта книга отличается от большинства книг,знакомящих чителей с компьютерами. Она рассчитана на тех читатей, которые не желают останавливаться на сведениях для начинающих, а хотят узнать действительные возможности персонального компьютера фирмы "IBM".

Компьютер IBM/PC позволяет реализовать множество интереснейших возможностей, о которых будет рассказано в данной книге. Эта книга адресована всем, кого действительно интересует как работает IBM/PC и что можно делать с ее помощью, независимо от того, является ли читатель опытным профессионалом или начинающим пользователем компьютеров.

Книга позволяет не только понять как функционирует IBM/PC, но и обьясняет как можно его использовать на практике. Она предлагает дополнительный и более глубокий материал об IBM/PC для всех, кто в нем нуждается. Мы поможем Вам понять не только принципы

функционирования машины, Но и ее потенциальные возможности. Книга содержит множество советов и практических рекомендаций, касающихся программирования для IBM/PC. Но не следует считать эту книгу руководством по программированию для IBM/PC, в основном она посвящена возможностям самого компьютера.

1.1. Краткий обзор содержания книги

Эта книга в основном посвящена программам (или программному обеспечению), поскольку именно они реализуют большинство возможностей компьютера. Но помимо этого необходимо рассмотреть и аппаратные средства компьютера - этому посвящена глава 2.

В главе 3 рассматривается функционирование ценительного узла компьютера - собственно микропроцессора.

Операционная система PC-DOS кратко описывается в главе 4. В этой главе приведены пояснения принципов работы ДОС и в качестве дополнительного материала описаны программные средства, обеспечивающие доступ к различным возможностям ДОС.

Далее, в главе 5, рассматривается организация внешней памяти и способы хранения данных на гибких магнитных дисках. Примеры программ показывают способы расшифровки служебной информации об организации хранения данных на дискете, которая обычно скрыта от пользователя.

Глава 6 посвящена программному обеспечению, которое хранится в ПЗУ (постоянном запоминающем устройстве) IBM/PC. В этой главе показано как можно пользоваться этими программами. Этот материал подготавливает почву для изучения глав с 7 по 11, в которых, шаг за шагом, описываются служебные программы, хранящиеся в ПЗУ. Каждая из этих глав сопровождается примером программ, позволяющих использовать те или иные возможности компьютера.

Глава 7 описывает доступ к процедурам работы с дискетами. Главы 8 и 9 описывают работу с дисплеем, соответственно в алфавитно-цифровом и графическом режимах.

Глава 10 посвящена использованию клавиатуры, а глава 11 подводит итоги рассмотрения всего предыдущего материала.

Приложения этой книги включают краткий глоссарий компьютерной терминологии, введение в Паскаль и описание интерфейса между программами, написанными на ассемблере, и программами на Паскале или другом языке высокого уровня.

1.2. Вспомогательные обучающие средства

Если Вы просто хотите узнать побольше об IBM/PC, то Вам не потребуется ничего, кроме этой книги. Однако, чтобы применить полученные знания на практике этого будет недостаточно, потребуется, как минимум, сам компьютер.

Чтобы воспользоваться программами, приведенными в этой книге, потребуется IBM/PC с 64K байтами памяти и одним дисководом. Все программы могут работать как с монохромным дисплеем, так и с цветным графическим адаптером. Потребуется также операционная система ДОС и поставляемые вместе с нею средства, такие как программа DEBUG. Можно воспользоваться любой версией ДОС: исходной версией 1.00, неофициальной и временной версией 1.05, усовершенствованной версией 1.10 или ДОС 2.00.

Чтобы как можно полнее использовать все возможности, предоставляемые IBM/PC, Вам потребуется дополнительный пакет программ на гибких магнитных дисках, прилагаемый к этой книге. Средства доступа ко всем возможностям IBM/PC были разработаны специально для этой книги. Содержимое пакета описывается в приложении 5.

Вам не нужно будет использовать Макроассемблер фирмы "IBM" для использования представленных в книге процедур доступа, написанных на ассемблере. Все эти программы включены в пакет программ в виде готовых к использованию обьектных модулей. Однако, если Вам захочется внести изменения в ассемблерную программу, чтобы адаптировать ее к своим нуждам, то потребуется и ассемблер, и необходимо будет освоить язык ассемблера. Один из разделов этой книги представляет собой простое введение в использование ассемблера.

Чтобы использовать приведенные в книге программы на языке Паскаль либо сами по себе, либо в составе Ваших собственных программ, Вам потребуется компилятор языка Паскаль для IBM/PC.

И, наконец, Вам может потребоваться копия "Сервисных программ Нортона" (The Norton Utilities). Она включает программу

восстановления поврежденных дискет (FileFix), уничтоженных файлов (UnErase), модификации секторов на дискете (SecMod) и управления скрытыми файлами (FileHide и BatHide); реорганизации справочников файлов (DiskOpt и FileSort) и управления экранным режимом (Reverse, ScrAtr и Clear), а также несколько других полезных служебных программ.

1.3. Используемые программные средства, языки программирования и тексты программ

В этой книге будет приведено множество программ и мы сразу же приступим к рассмотрению первой из них, текст которой приведен в приложении 1-1. Эта программа написана на языке БЭЙСИК и выдает на экран все 256 кодов символов, отображаемых дисплеем IBM/PC. Многие пользователи, как это ни удивительно, никогда не видели всего набора символов, так что это представляет для них интерес. Просмотр сразу всех возможных символов может оказаться полезным для выбора из их числа таких символов, которые будут использоваться для организации специальных эффектов. Программа отображает символы в виде таблицы по 16 символов в каждом ряду. Если необходимо установить порядковый номер любого символа, то можно использовать функцию CHR$ языка БЕЙСИК. Так первая строка содержит символы от CHR$(0) до CHR$(15), а вторая CHR$(16)-CHR$(81). Шестнадцатиричные коды символов определяются по меткам строк и столбцов.

В примерах программ, приведенных в этой книге, используются три языка программирования: БЕЙСИК, Паскаль и язык ассемблера. Интерпретатор языка БЕЙСИК имеется в любом варианте IBM/PC, поэтому БЕЙСИК используется во всех случаях, когда им целесообразно воспользоваться. Однако БЕЙСИК не слишком хорошо приспособлен для решения серьезных задач, поэтому, в основном используется язык Паскаль.

Паскаль приобрел большую популярность в мире персональных компьютеров благодаря своим возможностям, компактности и надежности. У Паскаля имеются и очень сильные конкуренты - в первую очередь языки Си и Форс. Однако, для нас Паскаль имеет два важных преимущества. Во-первых, он проще и его легче изучить, чем языки Си и Форс и во-вторых, что наиболее важно, фирма "IBM" поставляет Паскаль для своих персональных компьютеров с самого начала. И коль скоро нам необходим какой-то общий язык общения, большая часть программ в этой книге написана на Паскале.

Если вы не знакомы с языком Паскаль, не отчаивайтесь, наиболее простой способ немного изучить этот язык заключается в чтении примеров, приведенных в книге. Они были специально написаны как можно доступнее и подобраны таким образом, чтобы постепенно обучать читателя. Кроме того, приложение 2 содержит краткое руководство по языку Паскаль, демонстрирующее его возможности и стиль программирования, поясняющее наиболее важные компоненты языка.

Я настоятельно рекомендую язык Паскаль всем, кто еще не выбрал, на каком языке программировать на IBM/PC.

Если Вы собираетесь использовать другой язык, то примеры программ на Паскале, приведенные в этой книге, все равно будут полезны. Они показывают приемы работы на IBM/PC, которые можно перенести и в другие языки программирования.

Многие из наиболее мощных и интересных возможностей IBM/PC можно задействовать только с помощью программ, написанных на языке ассемблера. В этой книге мы рассмотрим все эти возможности и способы их использования. Чтобы обеспечить Вам наиболее благоприятные возможности для их использования в книге приведен полный набор программ, написанных на языке ассемблера, которые представят Вам доступ ко всем возможностям IBM/PC и операционной системы ДОС.

В приложении 3 описываются способы подключения модулей, написанных на языке ассемблера. Потребности описания относятся к языку Паскаль, но они практически в том же виде могут быть применены к любому языку программирования, который использует стандартные механизмы вызова программ.

Специально для этой книги было написано небольшое число программ.Одни - с чисто иллюстративным назначением - чтобы продемонстрировать некоторые приемы работы. Другие программы предназначены для помощи в получении информации о Вашем компьютере. Третью группу составляют программные средства, обеспечивающие Вам доступ ко множеству важных возможностей, предоставляемых IBM/PC. По возможности листинги этих программ были включены в эту книгу , и в первую очередь, это относится к тем программам, изучить которые читателю наиболее полезно. Однако, некоторые из вспомогательных программ лучше не изучать, а непосредственно использовать, книга содержит тексты только тех программ, которые полезно изучить, а все остальные программы

включены в специальный пакет, записанный на дискете, прилагающейся к данной книге.

Этот пакет содержит все программы, листинги которых приведены в книге, и многие другие полезные программы. Содержимое пакета описывается в приложении 5.

Программы, написанные на языке ассемблера, записаны на дискете как в виде исходного текста, так и в виде готовых к использованию обьектных модулей. Если Вы захотите внести в эти программы какие-либо изменения, то Вам пригодится исходный текст. Для использования готовых программ Вам не нужно изучать язык ассемблера и использовать ассемблер. Программы на Паскале также представлены и в виде исходного текста, и в виде готовых, скомпилированных модулей, которые готовы к использованию.

В пакет программ включена одна вспомогательная программа, DiskLook. Эта программа позволяет просматривать всю иформацию, хранящуюся на гибких магнитных дисках. Она может вывести перечень файлов, упорядоченный по именам, датам или размерам или имена всех файлов, которые были удалены. Программа DiskLook позволяет просмотреть распределение дискового пространства, показывает расположение любого файла и и позволяет считывать данные с любого места на дискете.

1.4. Три пересекающиеся области

Хотя не все это замечают, но появление IBM/PC привело к возникновению трех пересекающихся областей интересов. Чтобы это стало очевидным, должо пройти определенное время, но читатель должен знать о них и об их связи с данной книгой.

Появление персонального компьютера фирмы "IBM" было, как ни странно это звучит, появлением первого IBM/PC-подобного компьютера; первого, но не единственного. Кроме того, появилась операционная система фирмы "Майкрософт", MS-DOS; версия этой операционной системы для IBM/PC получила название PC-DOS, хотя можно встретить и названия IBM-DOS и просто ДОС.

Отсюда и возникают три области интересов. Во-первых, это интерес к самому компьютеру IBM/PC. Далее, это интерес к компьютерам в большей или меньшей степени имитирующим IBM/PC и, наконец, это интерес к семейству компьютеров, использующих операционную систему MS-DOS.

Между этими областями много общего, так что любая книга, посвященная одной из областей, содержит обширный материал и по остальным. Эта книга посвящена первопричине появления всех этих областей интереса - персональному компьютеру фирмы "IBM", но большая часть того о чем пойдет речь, может пригодиться и тем, кто интересуется IBM/PC-подобными компьютерами и тем, кто интересуется семейством компьютеров, работающих под управлением операционной системы MS-DOS.

Время от времени, при изложении материала этой книги, когда можно провести такое разделение, я буду указывать, что относится, а что не относится к остальным двум областям.

1.5. Источники информации

В такой книге нельзя описать абсолютно все аспекты и подробности работы IBM/PC. Ниже приведен список наиболее важных и полезных источников информации, которыми Вы можете воспользоваться, если потребуются более подробные сведения.

Практически все сведения, приведеные в даной книге, извлечены из этих источников. В отличие от многих других персональных компьютеров IBM/PC сопровождался очень широким кругом открытых источников информации. Это произошло благодаря позиции фирмы "IBM", состоящей в том, чтобы машина была как можно более доступна разработчикам программного обеспечения и аппаратных расширений. В качестве автора этой книги, я не имел доступа к каким-либо особенным секретам, какие были бы недоступны читателям. Мне потребовалось только переработать общедоступную информацию, извлечь из нее наиболее существенное и выделить информацию, представляющую интерес для наиболее широкого круга читателей.

Если Вам потребуется более подробная информация, чем та которая содержится в этой книге, Вы можете обратиться к следующим источникам: наиболее богатый источник информации - это собственное техическое руководство по Персональному компьютеру фирмы "IBM", содержащее помимо технических подробностей, детальное описание ROM-BIOS, т.е., хранящейся в ПЗУ базовой системы ввода-вывода. Эти программы обеспечивают наиболее функциональные программные средства для управления IBM/PC. Приведен листинг BIOS на языке ассемблера, в котором можно разобраться только в том случае, если Вы хорошо знаете язык ассемблера микропроцессора 8086 фирмы "Интел". Тем не

менее, листинг системы BIOS представляет собой настоящий клад информации о служебных подпрограммах, доступных пользователю. Даже если Вы незнакомы с языком ассемблера, описания служебных подпрограмм и способ их организации помогут Вам понять организацию этого компьютера. (Приводятся только листинги системы BIOS, а листинг записанного в ПЗУ интерпретатора языка Бейсик (ROM-BASIC) не приводятся, хотя его можно получить с помощью команды DEBUG операционной системы, правда без комментариев.)

Кроме того, в руководстве приведены две очень удобные таблицы, содеращие все 256 кодов символов, используемых IBM/PC и функции специальных символов-атрибутов, управляющих цветом текста, отображаемого на экране IBM/PC.

Еще одинм полезным источником информации может послужить руководство, поставляемое вместе с операционной системой DOC. В приложениях к этому руководству можно найти полезную информацию о форматах дискет, соглашениях для служебных процедур ДОС, блоках управления файлами, приставках сегментов программ и так далее. Само руководство по ДОС несколько беднее, чем большинство наших источников информации, поскольку оно не содержит подробной технической информации о ДОС. Руководство содержит очень полезную вспомогательную информацию и некоторые рекомендации.

Чтобы лучше понять функционирование микрокомпьютера, на основе которого функционирует IBM/PC, можно обратиться к нескольким книгам по микропроцессорам 8086/8088 фирмы "Интел". Особенно полезными мне показались две книги. Наиболее доступно принципы организации и работы микропроцессора 8086 представлены в книге "The 8086/8088 Primer", написанной Стефаном П. Морзом (Hayden,1980). Более глубоко микропроцессор описывается в книге "The 8086 Book", Рассела Ректора (Osborne/McGrawHill,1980).

В качестве справочника по программированию на языке Паскаль можно воспользоваться стандартным руководством по компилятору языка Паскаль для IBM/PC. Это не слишком удачное введение в Паскаль, но только в этом руководстве описываются особенности конкретной версии языка Паскаль для IBM/PC. Чтобы изучить язык можно выбрать одну из книг по Паскалю для начинающих, которыми переполнены книжные магазины. Если же Вы обладаете хорошей подготовкой, достаточно прочитать руководство по компилятору языка Паскаль полностью. Практически все технические детали,

касающиеся использования версии языка Паскаль для IBM/PC либо подробно описаны, либо поясняются примерами. На мой взгляд, авторы этого руководства проделали большую работу, включив все необходимые рекомендации для тех случаев, которые они не могли детально пояснить. (Вы многое сможете узнать о языке Паскаль, внимательно изучив примеры, приведенные в этой книге, и прочитав руководство по языку Паскаль в приложении 2).

Как и в случае с языком Паскаль, рекомендации по программированию на ассемблере можно найти в руководстве по ассемблеру для IBM/PC. Однако, в этом руководстве Вы не найдете набора машинных команд. Он описывается в двух упоминавшихся выше книгах по микропроцессору 8086. Руководство по ассемблеру для IBM/PC очень плохо об'ясняет вопросы программирования и практически не содержит необходимой вспомогательной информации, но без него невозможно пользоваться ассемблером. (Приложение 3 к настоящей книге описывает методы об'единения программ, написанных на языках Паскаль и ассемблер, а также содержит ряд полезных советов, которые помогут Вам начать изучение языка ассемблера.)

1.6.Несколько замечаний о способе изложения материала

В этой книге я старался об'яснять все как можно проще, яснее и по возможности без противоречий. Если Вам встретится непонятный термин, загляните в краткий глоссарий терминов, приведенный в приложении 1.

Ниже приведен ряд замечаний. поясняющих способ изложения материала, использованный в этой книге. Сначала о представлении чисел.

Числа всегда будут записываться обычными десятичными цифрами, которые все мы изучаем еще в детстве. В тех случаях, когда это необходимо, будут также приводиться шестнадцитиричные числа, обозначаемые словом "шестнадцатиричное". Шестнадцатиричные числа удобно использовать при работе с компьютером, поскольку они представляют собой удобную сокращенную запись двоичных чисел. В отличие от десятичной системы, использующей десять цифр, в шестнадцатиричной системе цифр шестнадцать, обозначаемых от 0 до 9, далее следует A (со значением 10), B, C, D, E, и наконец, F (со значением пятнадцать). В этой книге 11 глав (шестнадцатиричное B); Декларация Независимости была подписана в 1776 году (шестнадцатиричное 6F0).

Вас может удивить запись чисел и в десятичном и в шестнадцатиричном виде, хотя бы потому, что ни Паскаль, ни Бейсик, ни ассемблер не требуют использования шестнадцатиричных чисел при программировании. Тому есть три причины. Одна состоит в том, что некоторые числа, например, адреса ячеек памяти, имеет больше смысла представлять в шестнадцатиричном виде. Другая связана с тем, что вспомогательная программа ДОС DEBUG использует только шестнадцатиричную форму представления. И, наконец, часть литературы по IBM/PC, в частности техническое руководство, широко использует шестнадцатиричную форму, причем даже без специальных оговорок. Чтобы облегчить Вам сопоставление информации этой книги с другими источниками, числа приводятся как в десятичном, так и в шестнадцатиричном виде.

Еще один элемент компьютерного жаргона, связанный с записью чисел, - это использование буквы "К". "К" соответствует числу 1024, так что 64К будет сооветствовать 64 раза по 1024 или 65536. Термин "К" широко используется, поскольку он является кратким обозначением числа, примерно равного одной тысяче - так что легко понять, что 64К примерно соответствует шестидесяти четырем тысячам. в двоичной системе значение К или 1024 представляется круглым числом - это два в десятой степени.

Иногда возникает необходимость обозначать символы их кодами, значения которых находятся в диапазоне от 0 до 255. В этих случаях используется способ записи, принятый в языке Бейсик, например, CHR$(65), соответствующий заглавной букве "A". На языке Паскаль это же можно записать как chr(65).

Рассмотрим теперь биты. Очень часто непонимание и путаница возникает при обращении к отдельным битам, образующим байты и слова. Различные источники описывают их по-разному, что еще более осложняет изучение. Наиболее часто используются три схемы, которые я сейчас поясню. Трудно сказать какая из них наилучшая - мы будем пользоваться самой простой. Если Вы собираетесь знакомиться с технической литературой по компьютерам, Вам необходимо знать три схемы.

Если записать все восемь битов образующих байт так, чтобы первым был "старший значащий", а последним "младший значащий" бит, то их можно будет пронумеровать от первого до восьмого. Такую последовательность можно назвать просто "по порядку". Такой способ записи мы и будем далее использовать как наиболее простой. Недостаток его заключается в том, что он не имеет

никакого смыслового значения. Другие два метода связаны с числовым "весом" каждого бита. По второму методу биты нумеруются справа налево, начиная с нуля. Это способ записи "по степеням" и каждый номер бита соответствует степени двойки. Такой способ записи наиболее распространен в технической литературе. И последний метод, который можно назвать "по значению", обозначает каждый бит значением, равным двум в степени, соответствующей этому биту.

Ниже приведена таблица, показывающая все три способа записи:

По порядку	По степени	По значению	Бит
1-ый	7	128	10000000
2-ой	6	64	01000000
3-ий	5	32	00100000
4-ый	4	16	00010000
5-ый	3	8	00001000
6-ой	2	4	00000100
7-ой	1	2	00000010
8-ой	0	1	00000001

Как уже упоминалось, мы будем просто нумеровать биты слева направо, по порядку. Остальные два способа, особенно нумерацию "по степени", можно встретить в различной литературе. Нумерация "по значению" удобна в тех случаях, когда необходимо связать числовые значения байта с его битами. Например, в программе можно установть 1-ый и 8-ой биты, присвоив байту значение 128+1. Каждый из этих способов нумерации битов может быть расширен от 8-битового байта до 16-битового слова, что также может Вам встретиться в технической литературе.

Приложение 1.1 Текст программы визуализации всех символов на экране дисплея (Бейсик).

```
1000  REM Листинг 1.1 -- Программа для отображения всех
1010  REM символов экрана. (C) Авторское право 1983,
1020  REM Питер Нортон
1030  GOSUB 2000 'ЗАГОЛОВОК
```

```
1040  GOSUB 3000 'ОПРЕДЕЛЕНИЕ ТИПА ДИСПЛЕЯ И УСТАНОВКА АДРЕСА
1050  GOSUB 2000 'СНОВА УСТАНОВИТЬ ЗАГОЛОВОК
1060  GOSUB 4000 'ФОРМИРОВАНИЕ ОКРУЖАЮЩИХ КОММЕНТАРИЕВ
1070  GOSUB 5000 'ФОРМИРОВАНИЕ МАССИВА ЭКРАНА
1080  GOSUB 6000 'ЗАВЕРШЕНИЕ РАБОТЫ И ВОЗВРАТ В ДОС

2000  REM Подпрограмма вывода заголовка
2010  KEY OFF : CLS : WIDTH 80
2020  REM
2030  PRINT "   Программы для книги "Персональный компьютер"
2040  PRINT "   фирмы "IBM".(C) Авторское право 1983"
2050  PRINT "   Питер Нортон"
2060  PRINT
2070  PRINT "Программа 1-1: Демонстрация всех символов экрана"
2999  RETURN

3000  REM Подпрограмма проверки типа дисплея
3010  PRINT
3020  PRINT "Прежде чем продолжить работу сообщите,"
3025  PRINT "дисплей цветной-графический?";
3030  GOTO 3060
3040  PRINT
3050  PRINT "(ответ Д или Н)";
3060  ANSWER$ = INKEY$
3070  IF LEN(ANSWER$) < 1 THEN 3060
3080  IF LEN(ANSWER$) > 1 THEN 3040
3090  SEGVAL! = 0
3100    IF  MID$(ANSWER$,1,1)  =  "Y"  THEN  SEGVAL!  =  &HB800
'Цветной
3110    IF  MID$(ANSWER$,1,1)  =  "y"  THEN  SEGVAL!  =  &HB800
'Цветной
3120    IF  MID$(ANSWER$,1,1)  =  "N"  THEN  SEGVAL!  =  &HB000
'Монохр.
3130    IF  MID$(ANSWER$,1,1)  =  "n"  THEN  SEGVAL!  =  &HB000
'Монохр.
3140  IF SEGVAL! = 0 THEN 3040
3150  DEF SEG = SEGVAL!
3999  RETURN

4000  REM  подпрограмма для формирования сопроводительных
4005  REM  сообщений
4010  FOR HEX.DIGIT% = 0 TO 15
4020    LOCATE 6,HEX.DIGIT% * 3 + 1
4030    PRINT HEX$(HEX.DIGIT%)
```

```
4040    LOCATE HEX.DIGIT% + 8,8
4050    PRINT HEX$(HEX.DIGIT%)
4060  NEXT HEX.DIGIT%
4070  LOCATE  ,,0
4999  RETURN

5000  REM  подпрограмма форомирования массива экрана
5010  FOR ROW% = 0 TO 15
5020   FOR COL% = 0 TO 15
5030    POKE(ROW% + 7) * 160 + COL%*6 + 26, COL% + ROW% * 16
5040   NEXT COL%
5050  NEXT ROW%
5999  RETURN

6000  LOCATE 25,1,1
6010  PRINT "Нажмите любую клавишу для возврата в ДОС..."
6020  IF LEN(INKEY$) = 0 THEN 6020 'ожидание нажатия клавиши
6030  CLS
6999  SYSTEM

9999  REM Конец листинга программы 1-1
```

ГЛАВА 2. АРХИТЕКТУРА ТЕХНИЧЕСКИХ СРЕДСТВ

[Оглавление]

В настоящей главе мы рассмотрим физическую организацию персонального компьютера фирмы IBM, его периферию и принципы сопряжения отдельных компонентов. Поскольку предлагаемая книга не является руководством по организации технических средств, мы не будем рассматривать эти вопросы слишком детально. Приведенных сведений, однако, достаточно, чтобы понять принципы работы IBM/PC ,состав дополнительного оборудования, подключаемого к компьютеру и особенности его взаимодействия с остальной системой.

Компьютер во многом напоминает автомобиль. Он также состоит из множества отдельных частей и для того,чтобы им пользоваться (так же как в случае с автомобилем) не обязательно знать назначение каждой детали. Однако, если Вы хотите использовать всю систему, необходимо иметь хотя бы приближенное представление о том как функционируют отдельные части. Как раз этому и посвящена настоящая глава.

2.1. Микропроцессор-центральный уэел персонального компьютера

Центром вычислительной системы является ее процессор. Это основное звено, или "мозг" компьютера. Именно процессор обладает способностью выполнять команды, составляющие компьютерную программу. Персональные компьютеры строятся на базе микропроцессоров, выполняемых в настоящее время на одном кристалле или "чипе". Микропроцессор, использованный в IBM/PC, был разработан и создан фирмой "Интел", начавшей производство микропроцессоров много лет назад.

В IBM/PC используется микропроцессор 8088 фирмы "Интел", который практически полностью идентичен микропроцессору 8086 той же фирмы. Эти микропроцессоры (8086 и 8088) выполняют одни и те же команды, так что с точки зрения программирования они обладают функциональной эквивалентностью (различия между ними заметны только конструктору и мы остановимся на них чуть позже). Все, что касается программирования для микропроцессора 8086 относится и к микропроцессору 8088.

Принципиальное отличие IBM/PC от персональных компьютеров предыдущего поколения заключается в использовании 16-разрядного микропроцессора. До появления IBM/PC наиболее популярные персональные компьютеры строились на базе 8-разрядных микропроцессоров, таких как 6502 (фирмы "Моторолла", который использован в персональном компьютере "Эппл-2", 8080 (фирмы "Интел") или Z80 (фирмы "Зайлог"). Различия между 8- и 16-разрядными микропроцессорами весьма существенны, и их трудно сформулировать одной фразой. Любопытно,что наименее существенное различие дало этим микропроцессорам их названия: 8-разрядные процессоры могут манипулировать данными, состоящими из 8 бит, а 16-разрядные процессоры могут работать и 16-разрядными данными. Оба типа процессоров позволяют добиться одинаковых результатов, так что в этой части различие между ними не слишком значительно. Основное преимущество 16-разрядных процессоров перед их 8-разрядными предшественниками заключается в значительном повышении их быстродействия, мощности и удобства их набора команд (причем операции с 16-разрядными числами составляют лишь часть этого набора). Кроме того (и это самое главное) существенно увеличивается объем адресуемой памяти. Большинство 8-разрядных процессоров может использовать не более 64К памяти, что значительно уменьшает возможности эффективного использования персональных компьютеров. Процессор 8088,

используемый в IBM/PC, позволяет адресовать 1024К или свыше одного миллиона байт памяти. Может,разумеется,возникнуть потребность в памяти большего объема, но, для большинства практических нужд, одного миллиона байт вполне достаточно.

Таким образом, мы установили наиболее важное различие между процессором 8088 и 8-разрядными процессорами персональных компьютеров предыдущего поколения: объем адресуемой памяти больше не является жестким ограничением для задач, которые можно решать с помощью персонального компьютера.

В чем же состоит различие между микропроцессором 8088 и его "старшим братом" - микропроцессором 8086? Функционально они одинаковы - выполняют одинаковые команды, используют одни и те же данные, исполняют одинаковые программы. Отличаются они лишь способом обмена данными с внешней средой. Процессор 8086 работает с периферийным оборудованием, которое может параллельно обрабатывать 16-разрядные данные, а процессор 8088 может обмениваться только 8-разрядными словами. Таким образом, различие между процессорами 8088 и 8086 состоит в ширине внешней шины данных - процессор 8086 пересылает 16-разрядные, а 8088 - восьмиразрядные данные. Это дает основание утверждать, что микропроцессор 8088 не является в полном смысле 16-разрядным. Такое утверждение отчасти соответствует истине, но только отчасти. Внутреняя архитектура 8088 - 16-разрядная, но он не использует внешнюю 16-разрядную шину данных. Подробнее о шине данных будет идти речь в параграфе 2.3.

В практическом плане указанное различие между микропроцессорами 8086 и 8088 имеет два аспекта. Во-первых, при передаче более чем одного байта данных, процессор 8086 работает в два раза быстрее. Это не означает, что он выполняет всю работу вдвое быстрее, поскольку ожидание передачи данных занимает только часть времени работы и, кроме того, в некоторых случаях требуется передавать только 8 бит. Однако, в тех случаях, когда процессор ожидает передачу большого объема данных, 8086 тратит на ожидание меньше времени и, следовательно, выполняет работу быстрее.

Второй аспект заключается в проектировании схемы соединений и выборе компонентов. Восьмиразрядные цепи проще проектировать и в настоящее время имеется множество недорогих и очень надежных восьмиразрядных компонентов. Таким образом, воспользовавшись микропроцессором 8088, фирма "IBM" упрстила

свой персональный компьютер и уменьшила его стоимость ценой незначительного уменьшения скорости вычислений.

2.2. Краткие сведения об остальных компонентах компьютера

Для того чтобы микропроцессор мог работать, необходимы некоторые вспомогательные компоненты, подобно тому как одного двигателя недостаточно, чтобы заставить автомобиль двигаться.

Многие части автомобиля могут переноситься с одной модели на другую - для персональных компьютеров это еще более обычное дело. Лишь немногие компоненты IBM/PC были специально разработаны для нее, - фактически большая часть системы составлена из стандартных компонентов, начиная с микропроцессора 8088 фирмы "Интел". Особенность персонального компьютера фирмы "IBM" состоит в оригинальном способе организации известных компонентов в единую функционирующую систему. Электронная промышленность представляет разработчикам компьютеров большой набор необходимых стандартных компонентов, задача разработчика заключается в том, чтобы объединить их нужным образом.

Такое описание процесса разработки компьютеров может породить мысль, что разрабатывать компьютеры очень просто и, что все они очень похожи друг на друга. Но это настолько же верно, как то, что работа писателя заключается в выборе слов из словаря. Персональные компьютеры, подобные IBM/PC, действительно в значительной части состоят из стандартных компонентов, однако, главное заключается в способе их обьединения.

Составляющие IBM/PC можно рассматривать с трех различных точек зрения: по тому где они размещаются, как они функционируют, и как они взаимодействуют друг с другом. Рассмотрим вопрос пространственного размещения этих составляющих.

Физически составляющие IBM/PC можно разделить на компоненты системного блока и компоненты блока расширения. Все основные платы, входящие в состав любой модели IBM/PC, размещаются в большом блоке, получившем название системного. (Компьютеры фирмы "Эппл" используют более цветистое название для аналогичного блока - материнский блок.) Системный блок включает все необходимые компоненты, позволяющие компьютеру работать без каких-либо дополнений. Здесь находятся микропроцессор,

первые 64К памяти и "встроенные" программы, такие как интерпретатор языка Бейсик, записанный в микросхемах ПЗУ. Большая часть компонентов, описанных в следующем параграфе, также находится в системном блоке. На рис.2.1 показана упрощенная схема системного блока (показаны наиболее существенные детали).

Системный блок расположен в основании IBM/PC и заключен в корпус. Он занимает весь корпус в длину и примерно две трети в ширину. Если открыть корпус компьютера, то внутри, на дне корпуса можно увидеть системный блок. Если взглянуть на него со стороны задней стенки корпуса,можно увидеть в центре большой элемент схемы IBM/PC - микропроцессор 8088.

Правая часть системного блока находится под левым дисководом, а место слева от системного блока свободно - оно предназначено для размещения блоков расширений. В левом углу системного блока имеется пять свободных разъемов, предназначенных для подключения дополнительного оборудования, которое может быть введено в состав компьютера. Блоки расширения вставляются в эти разъемы, располагаясь над системным блоком.

Блоки расширения или карты, как их иногда называют - могут использоваться для обслуживания устройств, подключаемых к IBM/PC. Они могут использоваться для двух основных целей: для увеличения объема памяти и подключения дополнительных устройств. Если оборудование умещается на одной плате, то его можно разместить внутри корпуса IBM/PC. Если же оно не помещается в корпус, например, в случае с дисплеем, то внутри размещается только плата управления, соединяющаяся с оборудованием с помощью кабеля, который можно пропустить через отверстие в задней стенке корпуса. Каждому разъему расширения соответствует специальное отверстие в задней стенке корпуса, закрытое заглушкой, если оно не используется.

Системный блок разработан фирмой "IBM", а блоки расширения могут разрабатывать все желающие, при условии что они будут соблюдать основные правила, касающиеся размеров, электрических параметров соединений, теплового режима и так далее.

2.3. Функциональное назначение

Может Вам эти сведения и не пригодятся, но, видимо, будет интересно узнать как работают основные элементы схемы IBM/PC.

Если какой-либо элемент будет иметь значение для понимания материала последующих разделов книги, я буду это специально отмечать, чтобы Вы не подумали, что сведения, полученные в этом разделе можно просто забыть.

Сигналы синхронизации работы системы обеспечиваются генератором 8284A. Эти сигналы используются всеми элементами компьютера и задают длительность операций. С тактовым генератором связан таймер 8255A-5, использующийся для поддержки интерфейса накопителя на кассетной магнитной ленте и встроенного динамика. В главе 11 будет описано управление динамиком и мы увидим как "программировать" таймер для извлечения звуков.

Функционирование компьютерной системы основано на использовании прерываний, работа с которыми будет описана в следующей главе. Для организации работы системы прерываний используется микросхема 8259A.

Когда данные передаются внутри компьютерной системы, они проходят по общему каналу, к которому имеют доступ все компоненты системы. Этот путь получил название шины данных. Концепция шины представляет собой один из наиболее совершенных методов унификации при разработке компьютеров. Вместо того чтобы пытаться соединять все элементы компьютерной системы между собой специальными соединениями, разработчики компьютеров ограничили пересылку данных одной общей шиной. Данные пересылаются по шине в сопровождении специальных сигналов, обозначающих их назначение. Эта идея чрезвычайно упрстила конструкцию компьютеров и существенно увеличила ее гибкость. Чтобы добавить новый компонент, не требуется выполнять множество различных соединений, достаточно присоединить его к шине. Чтобы упорядочить передачу информации по шине используется контроллер шины 8288.

Все упоминавшиеся до сих пор элементы размещаются в системном блоке. Если взглянуть на основные блоки расширения, можно обнаружить еще несколько интересных компонентов. Имеется два типа адаптеров дисплеев для IBM/PC. Один из них предназначен для управления монохромным дисплеем фирмы "IBM" - для управления цветным графическим дисплеем (или простым монохромным дисплеем, который также может подключаться к цветному/графическому адаптеру). Хотя дисплеи этих двух типов работают по-разному и имеют различные характеристики, для

управления ими используется одна и та же микросхема - контроллер дисплея 6845 (фирмы "Моторола").

Для управления дисководами используется микросхема контроллера гибких дисков - PD765 фирмы "NEC" или ее эквивалент (расположена на плате адаптера гибких дисков). Изучая листинги BIOS, приведенные в техническом руководстве по IBM/PC, можно встретить таинственную ссылку на "NEC". Речь идет как раз о контроллере гибких дисков. Хотя мы не будем рассматривать такие подробности, можно упомянуть, что возможно непосредственное управление работой гибких дисков, путем выдачи команд контроллера. Эти команды описаны в техническом руководстве.

2.4. Использование разъемов расширения

Любые дополнительные устройства подключаются к IBM/PC с помощью одного из разъемов расширения, каждый из которых имеет 62 соединительных провода. Эти 62 линии позволяют передавать все сигналы, необходимые для управления любым оборудованием, которое может быть подключено к IBM/PC. Все линии работают параллельно, так что устройства можно подключать к любому из пяти разъемов. Любой сигнал, посылаемый одному из блоков расширения, передается и всем остальным, поскольку они подключены к параллельным линиям. Здесь имеет место расширение идеи общей шины данных: все блоки расширения используют общее 62-проводное соединение. называемое каналом ввода/вывода.

По характеру использования все линии можно разделить на четыре категории. Во-первых, восемь линий используются для подвода питания к блокам расширения с различными номиналами напряжений.

Далее, еще восемь линий используется для передачи восьми бит данных на/с шины данных. Все данные проходят по этой шине, независимо от направления передачи.

Еще двадцать линий предназначены для адресации. Когда данные передаются в память или считываются из нее, или данными обмениваются с внешним устройством, необходимо указать адрес, который может быть либо адресом ячейки памяти, либо номером устройства. При работе с памятью используются все 20 линий, это позволяет передать адрес одной из 1024К ячеек памяти. Для

устройств ввода/вывода используется только девять линий, что позволяет адресовать 512 различных устройств.

Остальные линии канала используются для передачи различных сигналов управления. Примерами таких сигналов могут служить команды чтения из памяти, записи в память или команды чтения/записи для периферийных устройств.

Каждое внешнее устройство, подключенное к разъему расширения постоянно ожидает сигналов канала ввода/вывода. Предположим, например, что выдана команда ввода, идентифицируемая сигналом на линии чтения по вводу/выводу. Когда это произойдет, все устройства будут читать шину адреса, который не относится к памяти компьютера (поскольку не была выдана команда работы с памятью).Если же выдана команда работы с памятью, то все устройтсва ввода/вывода будут игнорировать содержимое шины адреса. Поскольку запрашивалась операция ввода/вывода, каждое периферийное устройство проверит содержимое шины адреса. Если адрес на шине совпадает с адресом устройтсва, то оно начинает выполнять операцию. В противном случае никаких действий не производится.Таков принцип работы блоков расширения.

2.5. Что еще необходимо знать об аппаратных средствах

Есть еще несколько интересных подробностей, которые полезно знать о системном блоке IBM/PC.

Во-первых, внутри корпуса IBM/PC спрятаны два набора переключателей. Их называют переключателями конфигурации системы (они выполнены в виде корпуса с двумя рядами выводов, т.е. корпуса типа DIR). Установка этих переключателей указывает какое оборудование подключено к IBM/PC, например, количество дисководов, объем доступной памяти и т.д. Эти переключатели ничем реально не управляют - они используются только для удобства. После включения IBM/PC программы запуска считывают положение этих переключателей и затем устанавливают содержимоке определенных ячеек памяти в соответствии с их положением. Затем, если какой-либо программе необходимо узнать , какой объем памяти установлен, проверяется содержимое этих ячеек. (Хранение информации о положении переключателей в памяти очень удобно, поскольку появляется возможность, в случае необходимости, ее изменения. Таким образом, программа может изменить положение переключателей и, следовательно, как бы изменить список подключаемых устройств.)

Как видите, использование переключателей конфигурации системы "логическое" скорее чем "физическое". Изменение положений переключателей не отключает и не подключает никакие устройства, оно просто изменяет ту информацию, которую программы могут получить о конфигурации системы.

Далее, рассмотрим сопроцессор. Когда разрабатывался микропроцессор 8088, для него была предусмотрена способность выполнения обычных, целочисленных арифметических операций, но он не способен оперировать с числами с плавающей запятой или вещественными числами (в языке Бейсик это называется арифметикой обычной и двойной точности). Арифметические операции над числами с плавающей запятой могут выполняться одним из двух способов. Первый и наиболее распространенный способ - программная реализация с помощью логических операций и целочисленных арифметических операций подрограмм, выполняющих вычисления и дающих результаты в форме с плавающей запятой. Второй способ основан на использовании специализированного сопроцессора.

Микропроцессор 8088 сконструирован так, что он позволяет использовать арифметический сопроцессор 8087 фирмы "Интел". Специализация процессора 8087 состоит в быстрой обработке чисел с плавающей запятой. Он может выполнять как обычные операции сложения, вычитания, умножения и деления, так и более сложные операции, такие как вычисление тригонометрических функцмй. Конструктивно заложенные в микропроцессор 8088 сигналы позволяют ему передавать работу сопроцессору 8087, а затем получать результаты работы. Чтобы использовать арифметический сопроцессор, необходимо иметь его в составе компьютера, а кроме того необходимы программы, которые могут выдавать специальные коды, необходимые для запуска сопроцессора 8087. Хотя в первой версии PC фирма IBM не включала в систему сопроцессор, гнездо для него предусмотрено в системном блоке. На рис.2.1. его можно увидеть в правом верхнем углу, рядом с гнездом микропроцессора 8088.

Конструкция микропроцессоров 8086/8088 предусматривает два основных способа повышения вычислительной мощности. Первый заключается в использовании сопроцессора 8087, для которого в IBM/PC предусмотрено специальное гнездо в системном блоке. Второй способ заключается в организации мультипроцессорного режима, при котором несколько обычных микропроцессоров совместно выполняют вычисления, распределяя нагрузку между собой. Фирма "IBM" не предусмотрела такой режим работы в

конструкции своего персонального компьютера. Еще один вспомогательный "чип" - процессор ввода/вывода 8089 позволяет повысить общую производительность системы на базе процессора 8086/8088, однако его использование также не предусмотрено в IBM/PC.

Рассмотрим теперь еще одно пустое гнездо в системном блоке IBM/PC. "Встроенное" программное обеспечение IBM/PC записано в микросхемах постоянного запоминающего устройтсва (ПЗУ), расположенных почти в центре системного блока, ближе к левой стороне. Как видно из рис.2.1 таких микросхем пять. Рядом с ними, с левой стороны имеется свободное гнездо, которое оставлено специально с целью добавления каких-либо программ. Назначение этого гнезда вызывает различные домыслы. Лично я вижу три возможных разумных предназначения. Во-первых, оно могло быть оставлено по соображениям надежности - если в какой-нибудь из записанных в ПЗУ процедур обнаружатся ошибки, исправление которых приведет к увеличению объема процедуры, то в это гнездо можно будет установить микросхему с записью этих изменений. Во-вторых, если "IBM" расширит номенклатуру поддерживаемых системой устройств, таких как жесткие диски большой емкости, тогда в это гнездо можно будет установить микросхему ПЗУ с записанными программами управления. Правда, блок расширения для любого нового устройтсва также может содержать все необходимые программы, так что действительной необходимости в дополнительном гнезде для этих целей нет. Третья, и наиболее вероятная, возможность связана с поддержкой сопроцессора 8087. В это гнездо можно установть ПЗУ с программами сопроцессора. Такие программы позволили бы работать с арифметическим сопроцессором 8087 "встроенному" интерпретатору Бейсика, а также программам, написанным на других языках, например, на Паскале, Фортране или Бейсике (с помощью компилятора).

2.6. Три аспекта рассмотрения архитектурных решений

Все о чем мы говорили до сих пор в этой главе относится только к оригинальной модели IBM/PC, т.е. к первому аспекту. На совместимые с IBM/PC компьютеры распространяется лишь часть информации, а вся остальная применяется для каждого конкретного компьютера. Совпадает обычно лишь одна деталь всех совместимых с PC компонентов - это вид разъемов для подключения блоков расширения. В этой части все PC - подобные компьютеры практически полностью повторяют IBM/PC.

В сфере больших компьютеров уже давно образовался определенный круг так называемых элементов совместимых по способу соединения. Такие элементы могут заменить части системы путем отключения исходного элемента и включения на его место заменителя. Такая замена может производиться как со стороны периферийных устройтсв компьютера, так и со стороны процессора.

То же самое происходит и с IBM/PC. В месте любого соединения можно обнаружить конкуренцию между различными вариантами элементов замены по обе стороны соединения. Постоянным остается только формат разъема - его изменить нельзя. Таким образом, общим для всех компьютеров, совместимых с IBM/PC, будет формат разъема для подключения блоков расширения.

Материал, рассматривавшийся в начале этой главы, к третьей области интересов - компьютером, использующих различные версии операционной системы MS-DOS, - непосредственного отношения не имеет. Однако, в той части, где речь пойдет о значении дополнительного оборудования, подключаемого к IBM/PC, все сказанное об IBM/PC в равной степени относится и к другим персональным компьютерам.

2.7. Возможности комплдектации IBM/PC

В этом параграфе речь пойдет о различных возможностях комплектации IBM/PC дополнительными периферийными устройствами и о том, как это сказывается на рабочих характеристиках. Не все понимают значение периферийного оборудования. Дескать, IBM/PC и есть IBM/PC, независимо от того, что к нему подключено. Однако, для практического использования решающее значение имеет то оборудование, которое подключено к компьютеру.

По моему мнению существует три или четыре достаточно сильно различающихся персональных компьютера фирмы "IBM".

Речь идет об одной модели компьютера с различными комплектами периферийных устройтсв, которые определяют область применения компьютера.

ИГРУШКА. Это IBM/PC без дисковой памяти, так называемая "кассетная система". Такая конфигурация, по моему мнению, никогда и никем не воспринималась всерьез и, вероятно, не имела права на существование. Я убежден, что ее появление связано

только с желанием снизить цену базовой модели, чтобы не отпугнуть потенциальных покупателей. Такой комплект вряд ли можно использовать для решения сколько-нибудь серьезных задач, так что усеченный вариант мощной машины становится похож на игровую машину типа "Атари" или "Маттель". Доказательством бесполезности такого варианта может служить отсутствие программ на кассетах. Единственная возможность использования этой системы состоит, вероятно, в подготовке и обучении пользователей перед переходом к работе с более серьезным оборудованием.

Следующие два варианта комплектации IBM/PC являлись основными вариантами компоновки IBM/PC в начале ее применения. На их примере хорошо иллюстрируется положение о том, что в некотором смысле существует не один компьютер IBM/PC с различным периферийным оборудованием, а несколько существенно отличающихся друг от друга компьютеров IBM/PC.

ДЕЛОВОЙ КОМПЬЮТЕР. Это IBM/PC с монохромным дисплеем, желательно фирмы "IBM". Естественно, эта машина должна иметь два дисковода, устройство печати и, возможно, еще какое-то оборудование. Но отличает этот компьютер, прежде всего, дисплей. Графика здесь не требуется, вся работа ведется в алфавитно-цифровом (текстовом) режиме. Использолвание дисплея фирмы "IBM" очень облегчает чтение текста на экране. Сфера применения этого компьютера - в основном учреждения. Экран монохромного дисплея фирмы "IBM" не утомляет глаза, даже если смотреть на него целый день. С помощью этого компьютера можно выполнять деловые расчеты, формировать планы (например, с помощью программы "VisiCalk") или готовить документы с помощью текстовых процессоров.

ГРАФИЧЕСКАЯ МАШИНА. Это IBM/PC оборудованный цветным графическим дисплеем и остальными периферийными устройствами, включая дискеты. Этот компьютер позволяет работать с графическими изображениями, правда ценой пониженной различимости символов. Использование этой машины труднее описать простыми категориями, но они могут использоваться в инженерном и архитектурном черчении, деловой графике, играх и мультипликации.

Я утверждаю, что "деловой компьютер" и "графическая машина" являются различными компьютерами, поскольку они предназначены для совершенно разных целей. И, что еще более важно, пограммные продукты для них тоже совершенно различны. Ясно, что деловые программы, выдающие текстовые результаты, смогут

работать и на компьютерах с цветными графическими дисплеями, однако, это будет более утомительно для глаз. Программы, написанные для графического дисплея, определенно не будут работать на деловом компьютере.

Такое частичное разделение между предполагаемыми вариантами использования деловых систем IBM/PC делает их практически разными машинами, со своими рынками и своими кругом пользователей. Но все же, все системы IBM/PC имеют очень много общего в оборудовании, программных средствах и интересах пользователей. Я так подробно остановился на разделении деловых и графических систем, чтобы Вы лучше представляли себе сам факт существования такого различия.

МАШИНА С ВИНЧЕСТЕРОМ. Это IBM/PC, к которой подключен диск большой емкости с высокой скоростью доступа. Такой диск часто называют "жестким" (чтобы отличать его от "гибких" дисков) или "винчестером" (по кодовому наименованию одного из ранних проектов фирмы "IBM", в рамках которого разрабатывалась технология, реализованная в этих накопителях). Отличительной чертой такого компьютера является наличие внешней памяти большого объема, позволяющей решать множество задач: хранить всю деловую информацию небольшой компании, хранить регистрационные записи пациентов врача или хранить базу данных научного работника. Короче говоря, IBM/PC, оборудованный винчестером, обладает возможностями оперативной работы с большими массивами данных.

Не требуется большого опыта работы с IBM/PC, чтобы установить, что основным сдерживающим фактором быстродействия, возможностей и полезности этого компьютера является хранение данных на диске. Быстродействие микропроцессора 8088, использующегося в IBM/PC, для большинства задач достаточно велико. (Если какие-либо программы на Ваш взгляд работают недостаточно быстро, то это скорее всего связано с недостаточно эффективной реализацией. Наиболее общий пример - это программы на Бейсике, выполняемые в режиме интерпретации.)

Объем памяти IBM/PC также вполне достаточен и, во всяком случае, значительно превышает характеристики песональных компьютеров предыдущего поколения. Большинство первых персональных компьютеров имело всего 64 К памяти, в то время как объем памяти IBM/PC может превышать 256К. (Часть памяти используется в качестве экранной памяти, хранения интерпретатора Бейсика и управляющих программ системы BIOS).

Скорость и емкость внешней памяти на гибких дисках ограничивают возможности IBM/PC. Снять эти ограничения позволяют жесткие диски винчестерского типа. Обычно емкость винчестера лежит в пределах от 2 до 25 миллионов байт, что эквивалентно емкости 12 - 150 односторонних дискет. Кроме того, скорость доступа у винчестера намного выше, чем у дисководов с гибкими магнитными дисками.

Таким образом, с подключением винчестера появился четвертый и, вероятно, наиболее интересный вариант IBM/PC. Мы имеем все основания считать оснащенный винчестером IBM/PC отдельной версией компьютера, поскольку его возможности на порядок выше, чем у остальных систем. Винчестеры позволяют хранить базу данных значительного объема или всю деловую документацию компании. Винчестеры превращают персональные компьтеры либо в законченные деловые машины, либо в мощные средства доступа к данным. Таким образом, винчестеры существенно увеличивают возможности IBM/PC.

Когда появился первый вариант IBM/PC, фирма "IBM" не поставляла и не предусматривала подключение винчестеров к IBM/PC, но изготовители дисков быстро разработали контроллеры и программы для подключения винчестера к IBM/PC. Однако, можно ожидать, что для полной реализации потенциала IBM/PC винчестеры (или другие эквивалентные устройства памяти большой емкости с быстрым доступом) должны стать составной частью большинства компьютеров IBM/PC.

Прежде чем завершить рассмотрение важности комплектации IBM/PC периферийным оборудованием, необходимо упомянуть адаптер последовательной связи. Адаптер связи настолько обычный элемент оборудования, что о нем редко вспоминают. Однако, наиболее перспективный путь использования персональных компьютеров заключается в организации вычислительных сетей, которые без адаптера связи создать невозможно. Таким образом, адаптер связи может оказаться наиболее важным дополнительным элементом оборудования.

ГЛАВА 3. КАК РАБОТАЕТ IBM/PC

[Оглавление]

В этой главе мы рассмотрим работу "мозга" IBM/PC - микропроцессора 8088 - как он выполняет вычисления, использует память и общается с внешним миром. В этой главе

поясняются такие понятия как прерывание, стек и порт.

Практически весь материал этой главы полностью относится к уже упоминавшимся нами трем областям интересов. Это объясняется тем, что речь в основном будет тдти о функционировании микропроцессора 8086/8088, общего для IBM/PC, компьютеров, работающих под управлением операционной системы MS-DOS.

3.1. Память,часть 1: что это такое и как осуществляется чтение из памяти

Одним из основных элементов компьютера, позволяющим ему нормально функционировать, является память. Внутренняя память компьютера - это место хранения информации, с которой он работает. Внутренняя память компьютера является временным рабочим пространством; в отличие от нее внешняя память, такая как файл на дискете, предназначена для долговременного хранения информации. Информация во внутренней памяти не сохраняется при выключении питания.

По аналогии с конторской работой можно назвать микропроцессор конторским служащим, а память компьютера - рабочим столом: пространство его используется временно, для выполнения работы.

Память компьютера организована в виде множества ячеек, в которых могут храниться значения; каждая ячейка обозначется адрсом. Размеры этих ячеек и, собственно, типы значений, которые могут в них храниться, отличаются у разных компьютеров. Некоторые старые компьютеры имели очень большой размер ячейки, иногда до 64 бит в каждой ячейке. Эти большие ячейки назывались "словами". Супер-копьютеры Крей и компьютер Юнивак ориентированы на работу со словами.

Трудность работы со словами большой длины заключается в том, что обычно программы работают не с целыми словами, а с их частями. Поэтому большинство современных компьютеров, и в том числе все персональные компьютеры, используют значительно меньший размер ячейки памяти, состоящей всего из 8 бит или "байта": Байт - это очень удобная единица информации, отчасти потому что он позволяет хранить код одной буквы алфавита или одного символа. Поскольку символ занимает в точности один байт, термины "байт" и "символ" часто используются в одном и том же смысле.

Так как IBM/PC использует ячейки памти длиной восемь бит или один байт, в памяти могут храниться значения, которые можно выразить восемью битами. Это значения до двух в восьмой степени или 256. Смысл величины, записанной в ячейку памяти,

зависит от способа ее использования. Можно считать, что байт содержит код алфавитного символа - так называемый код ASCII. В то же время его можно рассматривать и как число. Все 256 возможных значений могут рассматриваться либо как положительные числа от 0 до 255, либо как числа со знаками в диапазоне от - 128 до + 127. Кроме того, байт может использовать как часть большого объема данных, например, строки символов или двухбайтного числа.

Для удобства манипулирования символьными данными компьютеру необходимо чтобы коды символов преобразовывались в
байтовые величины. Большинство компьютеров, включая IBM/PC, используют код ASCII, американсткий стандартный код дл[обмена информации. *Юольшинство компьютеров фирмы "IBM" используют другую схему кодирования символов, называющуюся EBCDIC; системы ASCII и EBCDIC организованы по разному, но перекодировка из одной системы в другую большого труда не представляет).

В коде ASCII числовые значения присваиваются всем обычно используемым символам, таким как буквы алфавита, строчные и заглавные, цифры, знаки пунктуации. Несколько кодов зарезервированы для управления, например, чтобы указать конец строки символов. Эти специальные управляющие коды в основном имеют значения от CHR$(0) до CHR$(31). Использование этих специальных кодов IBM/PC имеет ряд особенностей, которым посвящено приложение 4.

Код ASCII - это семибитовый код, имеющий 128 возможных значений кодов. Стандартный код ASCII обычно использует первые 128 из 256 возможных значений, помещающихся в байте. Остальные 128 могут использоваться для различных целей, образуя "расширенный набор" символьных кодов ASCII. Не существует никаких стандартов использования расширенных кодов и различные устройства компьютеров используют их по-разному. Дисплей IBM/PC использует расширенный набор языков, отличных от английского, различных математических символов, а также псевдографических элементов, которые можно использовать для рисования. Программа, приведенная в листинге 1.1, показывает все специальные символы IBM/PC, а подробнее об их организации сказано в главе 8.

Стандартное для IBM/PC матричное устройство печати MX-80 фирмы "Эндисон" преобразует коды символов расширенного набора ASCII в печатные формы, отличающиеся от изображений на экране. Если запустить программу 1.1, а затем использовать операцию PrintScreen (печать экрана) для копирования содержимого экрана на печатающее устройство, Вы сможете сравнить экранный и печатный эквиваленты одних и тех же

расширенных кодов ASCII.

Таблицы стандартных кодов ASCII и расширенных кодов ASCII для IBM/PC можно найти во многих местах. Одна из них приведена в конце руководства по Бейсику. Очень удобная форма таблицы приведена в приложении С к техническому руководству по IBM/PC.

До сих пор мы рассматривали побайтное использование памяти, однако, часто для более сложных значений, чем может уместиться в одном байте используется несколько байт вместе. Если необходимы строки символов, они сохраняются в соседних ячейках памяти, по одному символу на байт; первый слева символ записывается в первый байт, т.е., байт с наименьшим адресом.

Если требуется запомнить целое число больше одного байта, то оно записывается в несколько байт, также расположенных рядом. Наиболее распространенный формат использует 2 байта или 16 бит, что очень удобно для 16-разрядного процессора, такого как 8088. В терминах микропроцессора 8088 двухбайтное число называется словом. Многие команды 8088 срециально разработаны для работы со словами. Могут использоваться и более длинные форматы - трех-, четырехбайтные и длиннее - но они не так широко распространены как двухбайтные и для работы с ними нужны специальные программы.

Когда числа, состоящие из двух или нескольких байт, хранятся в памяти микропроцессора 8088, они размещаются в ячейках последовательно, начиная с младшего байта числа. Такой способ несколько непривычен для большинства специалистов, не имевших дела с микропроцессорами фирмы "Интел". Если Ваша программа работает с отдельными байтами в памяти, необходимо учитывать такой способ хранения.

Арифметический сопроцессор 8087 использует несколько специальных форматов, включающих четырехбайтовый целочисленный формат и три формата с плавающей запятой: двухбайтный, четырехбайтный и десятибайтный, а также десятичный формат с двадцатью десятичными цифрами. Микропроцессор 8088 непосредственно не использует эти форматы, но если к IBM/PC подключен арифметический сопроцессор 8087, эти форматы становятся как бы расширением набора форматов данных.

3.2. Память, часть 2: что такое адрес

Каждая ячейка памяти имеет адрес, который используется для ее нахождения. Адреса - это чмсла, начиная с нуля для первой ячейки увеличивающиеся по направлению к последней

ячейке памяти. Поскольку адреса - это те же числа, компьютер может использовать арифметические операции для вычисления адресов памяти.

Архитектура каждого компьютера накладывает собственные ограничения на величину адресов. Наибольший возможный адрес определяет обьем адресного пространства компьютера или то, какой обьем памяти он может использовать. Обычно компьютер использует память меньшего обьема, чем допускается его возможностями адресации. Если архитектура компьютера предусматривает небольшое адресное пространство, это накладывает суровые ограничения на возможности такого компьютера.

IBM/PC использует возможности адресации микропроцессора 8088 полностью. Адреса в 8088 имеют длину 20 бит, следовательно, процессор позволяет адресовать два в двадцатой степени байта или 1024 К.

Такое большое адресное пространствопозволяет свободно использовать ресурсы памяти для специальных целей, что мы увидим в следующем разделе. Перед этим необходимо разобраться в том, как 16-разрядный компьютер работает с 20-разрядными адресами и какие ограничения это может наложить на программы пользователя.

Большая часть арифметических операций, которые может выполнять микропроцессор 8088, ограничивается манипуляцией с 16-разрядными числами, что дает диапазон значений от 0 до 65.535 или 64 К. Поскольку полный адрес должен состоять из 20 разрядов, необходимо было разработать способ управления 20 разрядами. Решение было найдено путем использования принципа сегментированной адресации.

Если взять 16-ти разрядное число и добавить к нему в конце четыре двоичных нуля, то получится 20-ти разрядное число, которое может использоваться как адрес. Добавлением четырех нулей или сдвиг числа влево на четыре разряда фактически означает умножение числа на 16 и теперь диапазон значений будет составлять 1.024К. К сожалению, число с четырьмя нулями в конце может адресовать только одну из 16 ячеек памяти - ту, адрес которой оканчивается на четыре нуля. Все остальные ячейки, адреса которых оканчиваются на любую из остальных 16 комбинаций из четырех бит, не могут быть адресованы при таком методое адресации.

Для окончательного решения проблемы 20-разрядной адресации используются два 16-разрядных числа. Считается, что одно из них имеет еще четыре нуля в конце (выходящие за пределы разрядной сетки). Такое как бы 20-разрядное число называется сегментной частью адреса. Второе шестнадцатиричное число не сдвигается на четыре разряда и используется в своем

нормальном виде. Это число называется относительной частью адреса. Сложением этих двух чисел получают полный 20-разрядный адрес, позволяющий адресовать любую из 1.024 К ячеек памяти в адресном пространстве IBM/PC. Сегментная часть адреса задает ячейку с адресом, кратным 16, эта ячейка называется границей параграфа. Окончательное значение указывает конкретную ячейку на определеном удалении от границы параграфа.Рисунок 3.1 показыает как это делается.

Чтобы лучше усвоить этот момент, рассмотрим все еще раз. Полный 20-разрядный адрес задается двумя частями, каждая из которых представляет собой 16-разрядное число. Сегментная часть адреса обрабатывается так, как будто он имеет четыре дополнительных нуля в конце. Эта сегментная часть может относиться к любой части всего адресного пространства - но она может указывать только на шестнадцатиричную границу, то есть, на адрес, оканчивающийся на четыре нуля. Относительная часть адреса прибавляется к сегментной части, образуя полный адрес. Относительная часть адреса может задавать любую ячейку памяти, отстоящую от ячейки, указываемой сегентной частью, не более чем на 64К.

1-обычный 16-разрядный регистр адреса; 2-16бит (4 шестнадцатиричные цифры) 4 бита на шестнадцатиричную цифру; 3-сдвиговый регистр сегмента; 4-объединенные регистры сегмента адреса и сегмента; 5-дополнительный шестнадцатиричный ноль; 6-результат - 20-разрядный адрес

Хотя относительная часть адреса могла бы задавать только четыре последних цифры адреса, она принимает значения от 0 до без единицы 64К. Большая часть манипуляций с адресами связана с относительной частью адреса. Сегментная часть адреса фактически становится базовым адресом для рабочей области размером ,;К, которую позволяет адресовать относительная часть адреса.

Имеется удобный способ записи сегментированных адресов, использование которого программой DEBUG (описанной в главе 6) Вы еще увидите. Кроме того, он хорошо прослеживается в ассемблерных листингах, например в том, который приведен в приложении А к техническому руководсву фирмы IBM. Сначала записывается сегментная часть адреса, после нее следует двоеточие, а затем относительная часть адреса. Например, если сегментная часть адреса (в шестнадцатиричной форме) 2222, а относительная часть - 3333, то полный сегментированный адрес будет записываться как 2222:3333. Фактический 20-разрядный адрес будет в шестнадцатиричном виде иметь значение 25553,

получаемое таким простым сложением:

```
  22220
+
   3333
_________

  25553
```

(В конце этого раздела мы приведем несколько примеров работы с сегментированными адресами на Бэйсике и Паскале).

Для работы с сегментированными адресами микропроцессор 8088 имеет специальные регистры сегментов, предназначенные для хранения сегментной части адресов. Загрузив в регистр сегмента некоторое значение, можно адресовать следующие за ним 64К ячеек памяти . Без изменения значения в регистре сегмента компьютер может работать только с 64К байтами из общего адресного пространства в 1.024К. Путем изменения значения в регистре сегмента можно адресовать любую ячейку памяти.

Чтобы иметь возможность в каждый момент времени работать более чем с 64К памяти, в микропроцессоре 8088 предусмотрены четыре различных регистра сегмента, каждый из которых имеет особое назначение. Память компьютера используется для различных целей - часть ее занимает программа, другая часть используется для хранения данных, с которыми в данный момент работает программа. Поэтому два регистра сегмента выделены для программы и для данных. Для указания базового адреса программного или кодового сегмента используется регистр CS. Для указания сегмента данных используется регистр DS. Еще одна область памяти, используемая для специальных целей, называется стеком и ее адрес указывается регистром стека SS. И, наконец, для обеспечения дополнительных возможностей адресации имеется регистр дополнительного сегмента (или сегмента расширения), ES.

Когда программа подготавливается к выполнению, операционная система, такая как DOC, выбирает ячейки каких разделов будут использоваться для размещения кодовой части программ, данных и стека. В регистры сегментов CS,DS и SS заносятся адреса этих ячеек. При выполнении программы адреса в этих регистрах позволяют находить нужные ячейки памяти.

Следует понять, что эти регистры совсем не обязательнодолжны указывать на сегменты памяти, расположенные далеко друг от друга. Они могут указывать на любые ячейки, находящиеся далеко или близко друг от друга и даже располагающиеся в одном разделе. Если необходимо всего пару

тысяч байт для размещения программы и данных, кодовый сегмент и сегмент данных могут располагаться рядом. И хотя фрагменты кодового сегмента и сегмента данных используются при выполнении программы по-разному, 64К-байтные области, адресуемые соответствующими регистрами сегментов, могут отсекаться. На рис.3.2 показано как эти три сегмента - кодовый,данных и стека - могут использоваться и как области этих сегментов могут пересечься.

1-регистр сегмента; 2-указывает 64К памяти;
2-нам требуется; 3-16К кодовой части; 4-32К данных;
5-8К стека; 6-мы выделяем для них место в памяти и устанавливаем значения соответствующих регистров;
7-регистры обеспечивают доступ к большому обьему памяти, чем необходимо; 8-избыток

Если программа не изменяет содержимого своих регистров сегментов, то она может использовать только 64К данных и кодовую часть обьемом 64К. С другой стороны, если программа будет манипулировать содержимым регистров сегментов, то она сможет работать с данными любого обьема, вплоть до 1024К. Может использоваться любой из режимов работы, но на практике обычно фиксируют содержимое регистра сегмента данных и при необходимости изменяют содержимое регистра кодового сегмента. Такой способ организации работы поддерживается набором команд микропроцессора 8088, обеспечивающим очень удобный способ загрузки регистра кодового сегмента CS, путем использования команд FARCALL и FARRET.

Практические результаты такой организации выразились в том, что ДОС и языковые процессоры используют программные соглашения, которые позволяют увеличивать обьем программ практически неограниченно, в то время как их адресуемая область данных ограничена 64К. Вы легко заметите это ограничение, работая с Паскалем или компилятором Бейсика. Для "встроенного" интерпретатора Бейсика для IBM/PC собственно выполняемой программой является сам интерпретатор, а то что Вы считаете своей программой на Бейсике фактически является частью данных интерпретатора. Таким образом, для интерпретатора Бейсика суммарный обьем кодовой части и данных Вашей программы не должен превышать 64К, которые позволяет адресовать регистр данных DS.

Как Бейсик, так и Паскаль лишь до определенной степени

позволяют манипулировать сегментированными адресами. Вы можете непосредственно изменять содержимое регистров CS,DS,SS и ES -языковой процессор должен управлять этими регистрами, иначе все может совершенно запутаться. Однако, определенный способ использования в программах сегментированной адресации все же имеется.

Ниже приводится описание использования такого способа в Бейсике. Раздел сегмента может быть определен с помощью оператора DEFSEG. Некоторые языковые средства Бейсика, например, операторы PEEK и POKE, работают с адресами относительными к значению, заданному оператором DEFSEG.

Например, если взять упоминавшийся выше адрес 2222:3333, то на бейсике доступ к его значению может осуществляться следующим образом:

```
10  DEFSEG=$H2222  'установить  значение сегмента равным
                    шестнадцатиричных 2222
20 X=PEEK(&H3333)' выбрать значение, смещенное  на  3333
                    шестнадцатиричных
30 REM чисто для примера проверяем  в каком регистре за-
                    дан  код  буквы  и  если она в нижнем
                    регистре, то  переводим  ее в верхний
                    регистр
40IF(CHR$(X)>="a")AND(CHR$(X)<="z")THEN POKE&H3333,(X-32)
```

Таким образом, в программе на Бейсике доступ к любой ячейке памяти осущес твляется с помощью комбинации операторов DEF SEG и PEEK или POKE. Работа с ними встречается в листингах программ 1.1, 3.1 и 8.1, использующих такую возможность Бейсика для различных целей.

Паскаль разрешает аналогичный способ программного использования сегментированной адресации, правда в более унифицированном и гибком виде чем Бейсик. На Паскале можно определить переменную как сегментированный адрес,например:

```
VAR пример_адреса : adasmem,
```

а затем непосредственно задать его сегментную и относительную части ('.s' и '.r', соответственно):

```
пример_адреса.s : = # 2222;
пример_адреса.r : = # 3333;
```

Когда все эти присваивания выполнены, можно осуществлять

доступ к памяти с помощью указателя сегменитрованного адреса:

```
x : = пример_адреса^;
if(chr(x)> = "a") and (chr(x)< = "z") then
пример_адреса^ : = x - 32;
```

3.3. Организация памяти IBM/PC

Имея адресное пространство свыше миллиона байт IBM/PC позволяет более удобно и более гибко использовать память, чем большинство других, меньших персональных компьютеров. Весь объем памяти распределен довольно интересным образом. С одной стороны, резервирование определенных ячеек памяти может наложить ограничение на возможные применения компьютера. В IBM/PC резервируется несколько областей в верхних адресах памяти. Эти область имеют особое назначение, а все нижние адреса оставлены для свободного использования. Таким образом сочетаются достоинства использования зарезервированных блоков памяти и сохранения как можно большего объема памяти для свободного использования.

На рис. 3.3. показана простая схема распределения памяти. Верхняя четверть общего объема памяти, начиная с ячейки С 000 и до конца, практически всегда занимается постоянным запоминающим устройством или ПЗУ. (В бальнейшем все упомиющиеся адреса будут без специального напоминания даваться в шестнадцатиричном виде.) Фактически ПЗУ занимает только верхние 8К, начиная с адреса FE00, в которых находится система BIOS (подробно описанная в главе 6). Система BIOS в ПЗУ включает все основные служебные процедуры IBM/PC, в том числе тестовые программы, которые запускаются при включении компьютера.

Следующие (если идти сверху вниз) 32К используются для хранения интерпретатора Бейсика. Бейсик-ПЗУ начинается с ячейки F600 и заканчивается непосредственно перед BIOS. В главе 6 описывается интепретатор Бейсика в ПЗУ. В принципе, любая часть адресного пространства может быть занята постоянной памятью, но в соответствии с общими правилами организации памяти IBM/PC, адреса ПЗУ должны быть больше С000. Как упоминалось в главе 2, в системном блоке IBM/PC установлено пять микросхем ПЗУ и одно свободное гнездо для установки микросхемы. Эти пять микросхем содержат BIOS и Бейсик. Каждая микросхема имеет емкость 8К. Еще 8К можно добавить с помощью свободного гнезда и начальный адрес для

них будет равен F400, так чтобы они располагались непосредственно перед Бейсиком.

1-шестнадцатиричный адрес; 2-память, размещенная в системном блоке; 3-память,размещаемая в блоке расширения; 4-предел обычной памяти в 256К; 5-свободная часть; 6-зарезервирована для дисплеев; 7-монохромный дисплей; 8-свободна для использования дисплеями; 9-цветной графический дисплей; 10-область дисплейной памяти; 11-область возможного добавления ПЗУ; 12-Бэйсик-ПЗУ; 13-BIOS-ПЗУ; 14-нижние адреса; 15-верхние адреса

Ниже области ПЗУ располагается большой сегмент памяти, предназначенный для поддержки экранного режима дисплеев. Для хранения информации, отображаемой на экране, необходимо использоватьспециальную область памяти, которая может располагаться как внутри дисплея, так и внутри компьютера, с которым он соединен. В IBM/PC экранная память входит в общее адресное пространство компьютера (хотя конструктивно она размещается на плате расширения для дисплея). В главе 8 мы более подробно рассмотрим размещение и использование экранной памяти.

Блок экранной памяти начинается с адреса B000 и занимает 64К, вплоть до адреса C000. Этот блок делится на две части. Нижняя половина, начинающаяся с адреса B000, используется монохромным дисплеем. Верхняя половина, начинающаяся с адреса
B800, используется цветным графическим дисплеем. В главах 8 и 9 более подробно описывается структура и использование этой памяти.

Ни один из дисплейных адаптеров не требует и не использует все 32К, выделенные для них. Монохромный дисплей использует всего 4К, а цветной графический дисплей использует 16К. Оставшаяся часть памяти в настоящее время не используется, но может понадобиться для более совершенных дисплейных адаптеров.

Фактически, для дисплейных адаптеров выделено больше памяти, чем эти 64Кот B000 до C000. Блок в 64К, предшествующий им, также зарезервирован. В соответствии с документацией IBM/PC блок, начинающийся с адреса A000, резервируется двумя способами. Первые 16К, от A000 до A4000, зарезервированы совершенно таинсвенным образом, не имеется ни малейшего указания на то, для чего это сделано. Остальные 48К

этого блока от A400 до B000, входит в обалсть 112К, которая считается зарезервированной для экранной памяти. Таким образом, вся область экранной памяти располагается с адреса A400 до C000.

Можно предположить, что эта большая область, особенно 18К в блоке A000, будут использоваться каким-нибудь новым дисплеем с высокой разрешающей способностью, которому потребуется больше памяти чем для обычного монохромного или цветного графического дисплея. Непонятно только назначение первых 16К блока A000, не обозначенных как часть эуранной памяти.

Объем памяти, расположенной ниже адреса A000, составляет 64К, которык предназначены для обычного использования памяти компьютера. Первые 64К, до адреса 1000, располагаются на основной системной плате IBM/PC, а все остальные микросхемы памяти размещаются в блоках расширения. По сообщениям фирмы
"ИБМ", IBM/PC поддерживает всего 256К памяти, но этот предел связан только с тестами BIOS, которые при запуске компьютера проверяют такой объем памяти. Всегда можно подключить больше чем 256К памяти.

Вся обычная оперативная память, подключенная к Вашему компьютеру, располагается в нижних адресах общего адресного пространства. Вы можете подключить такой объем памяти какой Вам необходим в пределах, накладываемых зарезервированными адресами. Независимо от того, подключено ли к Вашему компьютеру 48К или 576К, они всегда размещаются начиная с адреса 0000.

Таким образом, оперативная память всегда занимает нижние адреса пространства, а постоянная память - в верхних адресах. Между ними располагается экранная память. Между всеми этими разделами могут быть промежутки - промежуток от конца оперативной памяти до начала экранной памяти и от конца экранной памяти до начала ПЗУ. Это неиспользуемая часть памяти IBM/PC.

Оперативная память временно используется работающими программами и их данными. Часть этой памяти, ее начальные адреса, используются для нужд самого микропроцессора, а другие небольшие фрагменты используются операционной системой
и интепретатором Бейсика. Оставшаяся память может использоваться любыми программами.

Самая нижняя часть памяти, начинающаяся с нулевого адреса и занимающая примерно 1500 байт, предназначена для хранения необходимой рабочей информации для компьютера. Первая часть этой области содержит вектора прерываний,

которые более подробно будут рассмотрены в разделе 3.5. После векторов прерываний располагается информация, необходимая для управляющих процедур BIOS, операционной систмы ДОС и интепретатора Бейсика, а также их рабочие ячейки. После зарезервированной области в нижних адресах памяти начинается рабочая область, в которую загружаются программы и где хранятся их данные.

Можно исследовать память IBM/PC и установить, какие блоки используются и где они размещены в адресном пространстве. Программы могут пытаться выполнять чтение и запись по любым адресам памяти. Можно предположить, что обращение к неподключенной памяти приведет к появлению сигнала ошибки, но на практике этого не происходит. Причина этого проста - работа микропроцессора 8088 с памятью заключается в обращении к каналу ввода/вывода, который описан в главе 2, и ожиданием результата. Даже если память не подключена, микропроцессор не замечает разницы.

Однако, можно достаточно просто проверить наличие или отсутствие памяти по определенному адресу. Один из методов заключается в чтении из различных облас тей памяти с последующей проверкой результатов. Непродолжительные эксперименты показывают, что при большинстве методов чтения, как через DEBUG, так и с помощью средств языка Паскаль, результатом всегда оказывается байт со всеми единичными битами; что соответствует шестнадцатиричному значению FF или CHR$(255).

Такой результат никого не должен удивлять - ведь при чтении несущетвующей памяти наиболее вероятно получить либо все нули, либо все единицы. По какойто причине, которая мне досих пор не понятна, интепретатор Бейсика всегда возвращает значение CHR$(233). Результаты не совсем однозначны, но в большинстве случаев Бейсик возвращает именно это странное значение, CHR$(233), при чтении неподключенной памяти.

Такое свойство можно использовать для проверки того, какая часть адресного простанства активна. Листинг 3.1. показывает простую программу на Бейсике, которая считывает по нескольку байт из каждого килобайта памяти и сравнивает их со значением CHR$(233). Если Вы запустите эту программу на своем компьютере, то она покажет какой объем памяти к нему подключен. Рисунок 3.4 показывает результат выполнения этой программы на IBM/PC, который я использовал при написании этой книги. Посмотрим о чем они говорят.

Во-первых, отметим, что были обнаружены три блока памяти - то есть, именно столько, сколько должно быть подключено к IBM/PC. Первый блок начинается с адреса 0 или самого начала памяти. Это обычная, оперативная память компьютера. В данном

конкретном компьютере первый блок имеет размер 576Кбайт. Сюда входят 64К в системном блоке и еще 512К в блоке расширения.

Второй блок активной памяти начинается с адреса B000. Это память, используемая адаптером монохромного дисплея. Поэтому можно сделать вывод, что данный компьютер работает с монохромным дисплеем, а не с цветным графическим адаптером. Заметим, что наша исследовательская программа считает, что эта память занимает 32К, в то время как в действительности монохромный адаптер включает включает всего 4Кбайта. Лишние 28К обнаруженные программой, можно, вероятно, отнести на счет побочных эффектов, связанных с работой схем, которые поддерживают реальные 4К памяти. Выполнение той же самой программы в системе с цветным графическим монитором также обнаруживают 32К памяти, хотя реально используется всего 16К.

После памти адаптера дисплея программа обнаруживает ПЗУ, занимающее верхние адреса памяти. В этом последнем блоке программа обнаруживает ровно столько байт, сколько в нем действительно имеется.

Программа 3-1: Найти и отобразить активную память
Потерпите - выполнение этой программы занимает 65 сек
(Адреса начала сегментов приведены в шестнадцатиричном виде)

```
Активная память начинается с  0
            заканчивается 8FFF 576К байт
                        (589824 байта)
Активная память начинается с  B000
            заканчивается B7FF 32К байт
                        (32768 байта)
Активная память начинается с  F000
            заканчивается FFFF 64К байт
                        (65536 байт)
Конец работы
```

Рис 3.4. Результат работы программы 3.1

Хотя программа 3.1 интересна и сама по себе, для исследования памяти Вашей систем, но она также может послужить хорошим примером тех возможностей, которые откроются перед Вами, когда Вы досконально изучите компьютер. Фирма "IBM" не сообщает, что Бейсик считывает из отсутствующей памяти именно значение 233, и никто не объяснял мне, как проверять память. Однако, имея интерес и некоторую настойчивость в исследованиях, я написал программу, приведенную в конце главы, листинг 3.1. Именно к такой работе

я Вас и призываю. Изучайте Вашу систему, исследуйте все детали - и тогда Вы приобретете глубокие познания о принципах работы IBM/PC.

Кстати, следует заметить, что эта программа может дать странные результаты, если используются платы памяти, изготовленные не фирмой "IBM". Это еще один признак, позволяющий больше узнать о Вашей системе.

3.4. Сверхоперативная память - регистры

Помимо памяти, для временного хранения данных микропроцессор 8088 использует еще и регистры, что существенно ускоряет работу. Сейчас мы кратко рассмотрим использование регистров.

Наиболее полное писание регистров можно найти в руководствах по системе команд и языку ассемблера микропроцессора 8076. В данном разделе содержится достаточно сведений для того, чтобы Вы не терялись, встретив упоминание названий регистров в любом контексте.

Микропроцессор имеет четыре шестнадцатиразрядных регистра общего назначения, называемых AX, BX, CX и DX. Каждый из них может быть разделен на два восьмиразрядных регистра, указанием старшей (H-high) или младшей (L-low) части полного (X) регистра. Таким образом, восьмиразрядные регистры называются AH, AL, BH, BL, CH, CL, DH и DL. Поскольку восьмиразрядные регистры непосредственно связаны с 16-разрядными, операция записи шестнадцатиричной величины ABCD в регистр AX эквивалентна записи величины AB в регистр AH и величины CD регистр AL.

Хотя все эти регистры имеют общее назначение, каждый из них имеет определенную специализацию. Например, если некоторая операция должна повторяться нес колько раз, то регистр содержит счетчик повторений.

Мы уже упоминали четыре регистра для сегментированной адресации:CD для кодового сегмента, DS для сегмента данных, SS для сегмента стека и ES для досполнительного сегмента. Относительная часть сегментирванного адреса может храниться в любом регистре и использоваться вместе с сегментной частью, которая должна находиться в одном из регистров сегментов. Текущий адрес в выполняемой программе образуется содержимым регистров IP (счетчик команд) и CS. Текущий адрес в стеке образуется содержимым регистра указателя стека (SP) и регистра сегмента SS.

Для адресации данных вместе с регистром сегмента DS могут использоваться разные регистры. Часто для этой цели

служит регистр DX. Два специальных регистра, DI и SI также используются свместно с регистром DS. При работе со строками байт, регистры SI и DI используются для хранения индексов исходной и приемной строк относительно регистра DS. Регистр указателя базы (BP) может спользоваться в качестве адреса относительно регистра SS.

3.5. На что необходимо обратить внимание - прерывания

Компьютер должен обладать способностью реагировать на события, происходящие вне его микропроцессора, например, чтобы воспринимать информацию, вводимую с клавиатуры. Существует два способа организации такой реакции.Один способ состоит в постоянном ожидании события. Такой способ называется "сканированием" или "опросом" и такой опрос может занимать большую часть времени компьютера. Другой способ позволяет компьютеру спокойно выполнять свою работу, пока не произойдет событие, требующее его внимания. Такой подход называется использованием "прерываний". использование прерываний позволяет наиболее эффективно организовать работу компьютера, поскольку время центрального процессора не расходится вхолостую на ожидание.

Какие примеры прерываний можно было бы привести? Например, прерывание возникает, когда Вы нажимаете клавишу на клавиатуре. Другое прерывание вырабатывает таймер, встроенный в IBM/PC. Таймер прерывает работу микропроцессора 8088 восемнадцать раз в секунду. Операционная система ведет отсчет времени дня просто подсчитывая эти прерывания таймера; время дня можно вычислить по количеству прерываний таймера после последней полуночи. Еще один вид прерываний формирует контроллер гибких дисков, сообщая процессору, что очередная дисковая операция наконец завершилась. Поскольку эти операции значительно медленнее чем работа самого микропроцессора, выгода состоит в том, что процессор может не ждать завершения дисковой операции и продолжать вычисления до тех пор, пока контроллер выработает прерывание, указывающее что операция завершена.

Основная идея прерываний заключается в том, что все требующее внимания процессора, должно сообщать ему о себе в форме прерывания. Использование прерываний дает огромные преимущества. Если ничто не требует внимания процессора, - а так обстоит дело большую часть времени, - то процессор не тратит времени на проверку наступления событий, требующих его вмешательства.С другой стороны, если возникает событие, требующее внимания немедленно, оно немедленно и будет обслужено, поскольку прерывания обычно обслуживаются сразу же

после их возникновения.

Иногда бывает необходимо, чтобы работа процессора не прерывалась, например, при выполнении какой-либо критичной операции. Для этого у микропроцессора 8088 имеется специальная команда, которая позволяет отложить обслуживание прерываний, запоминая их, и парная ей команда, восстанавливающая нормальный режим обслуживания прерываний.

Когда прерывания запрещаются запрос прерываний не теряется - он запоминается и будет обслуживаться, как только будут разрешены прерывания.

Обычно прерывания не запрещаются на сколько нибуть продолжительное время. Прерывания допустимо запрещать лишь на

очень короткие промежутки времени, необходимые для выполнения некоторых внутренних операций процессора, состоящих из небольшого числа команд. Типичным примером таких операций, которые не могут быть прерваны на полпути, может служить загрузка нового набора значений в регистры сегментов. Поскольку эти регистры необходимы для правильной работы любой программы, нарушение согласованности загрузки в них значений может привести к полной неразберихе, поэтому необходимо запретить прерывания на время загрузки в них новых адресов.

Чтобы привести небольшой пример того, к чему может привести запрет прерываний, рассмотрим две программы, названные BEEP и WARBLE. Если у Вас есть пакет программ, прилагающийся к этой книге, Вы найдете в нем обе эти программы, готовые к использованию. С помощью макроассемблера можно получить эти программы из ассемблерного листинга 3.2, приведенного в конце главы. Обе программы: BEEP и WARSLE, генерируют звуки с помощью динамика IBM/PC. (О том как генерируются звуки будет рассказано в главе 11.) Каждая из программ генерирует чистый непрерывный звук в одной тональности, но программа WARBLE не запрещает прерывания. Поэтому она прерывается таймером восемнадцать раз в секунду и хотя прерывания обслуживаются очень быстро, звук получается вибрирующим. Программа BEEP запрещает прерывания и дает непрерывный звук. Запустите обе программы и Вы почувствуете результат запрета прерываний.

Механизм перывания работает следующим образом: каждому из основных типов прерываний присвоен свой номер. Например, прерывание таймера имеет номер 8, гибкие диски, используют номер 14. В самом начале оперативной памяти IBM/PC хранится таблица с адресами программ, которые должны вызываться при возникновении различных прерываний. Эти адреса должны быть полными сегментированными адресами, так что они занимают по

два слова или по четыре байта каждый. Эти адерса иногда называются векторами прерываний. Прерывание с номером 0 имеет вектор, хранящийся в ячейке с нулевым адресом, прерывание ! имеет свой вектор в ячейке 4 и так далее. Когда происходит прерывание номер "X", вектор, хранящийся по адресу 4X, загружается в регистры адреса программы, т.е., регистры CS и IP,и компьютер начинает выполнять программу обслуживания прерывания, которая размещается по этому адресу.
Когда обработка прерывания заканчивается, программа обработки возвращает управление программе, которая выполнялась в момент возникновения прерывания, с помощью специальной команды IRET или "возврат из прерывания". Чтобы такой возврат мог быть выполнен, необходимо сохранить в стеке текущие адреса прогрвммы до загрузки в регистры CS и IP вектора прерывания.

Концепция прерывания была разработана для практического решения проблемы взаимодействия компьютера с внешним миром. Однако, тем не менее вскоре оказалось, что прерывания не менее полезны для организации внутренней работы компьютера. Это привело к разработке трех типов прерываний, которые получили названия аппаратных, логических и программных. Между ними нет принципиальной разницы, однако их использование разделит их на три отдельных категории.

Аппаратные прерывания мы уже рассмотрели - они вырабатываются устройствами, требующими внимания процессора. В IBM/PC таких прерываний на удивление мало. Во-первых, имеется так называемое немаскируемое прерывание, используемое для сообщения об отказе питания, оно имеет номер 2. Далее, прерывание 8 используется таймером, номер 9 - клавиатурой и 14 - контролером гибких дисков, всего четыре аппаратных прерывания. Имеется также семь зарезервированных номеров прерываний, 6, 7, с 10 по 13 и 15, которые могут быть испоьзованы в дальнейшем, если возникнет необходимость в дополнительных аппаратных прерываниях. Два из этих семи прерываний уже нашли свое назначение, прерывание 12 зарезервировано для адаптера связи, а прерывание 15 - для интерфейса устройства печати.

Логические прерывания формируются самим процессором 8088, когда он встречает какое-либо необычное условие. Таких прерываний предусмотрено четыре. Прерывание 0 возникает при попытке деления на ноль. Прерывание 1 используется для управления пошаговым режимом работы микропроцессора, при котором команды выполняются по одной. Это прерывание выставляется отладчиками для пошагового выполнения программ. Прерывание 3 вырабатывается командой установки "контрольных точек", которая также используется при отладке. Прерывание 4

формируется при возникновении условия переполнения, например, если результат арифметической операции не помещается в регистр. Таким образом, четыре логических прерывания распадаются на две пары: одна для арифметических операций (деление на ноль и переполнение) и вторая для отладки программ (шаговый режим и контрольные точки).

Наиболее интересны программные прерывания. Если программа должна воспользоваться услугами другой программы, она должна передать управление этой процедуре. Обычно это делается с помощью команды CALL. Для вызова процедуры программа должна знать ее адрес, а вызываемая процедура может не знать адреса вызывающей программы, поскольку механизм вызова автоматически генерирует адрес возврата, который будет использован вызываемой программой после завершения ее выполнения. Образно говоря, для перехода к подпрограмме и возврата из нее достаточно купить билет в один конец - обратный билет предоставляется бесплатно. Идея организации программных прерываний должна позволять свободный переход и в прямом и в обратном направлении, то есть иметь возможность вызвать подпрограмму и получить управление обратно после ее завершения, так чтобы ни одна из сторон не могла не знать о местонахождении (адресе) другой.

Программные прерывания обеспечивают такую возможность путем выработки прерывания самой программой. Например, если программе необходимо вычислить время дня, ей совершенно не требуется знать адрес программы подсчета времени - достаточно знать только, что программа подсчета времени дня запускается программным прерыванием 26.

Программные прерывания используются для вызова всех служебных функций, представляемых обычным пользователям. Эти функции включают все процедуры системы BIOS и ПЗУ и служебные
процедуры ДОС. Программные прерывания используются вместо непосредственных адресов по двум причинам. Наиболее важная причина состоит в обеспечении возможности изменения служебных процедур, вызываемых с помощью прерываний. Изменение процедуры обычно приводит к изменению ее размера и размещения в памяти. Если процедура вызывается с помощью прерывания, то использующие ее программы не требуется изменять, когда изменилась процедура.

Другая причина использования программных прерываний для вызова процедур связана с возможностью их замены. Рассмотрим пример. Как Вы узнаете из главы 8, служебные процедуры для обслуживания дисплея выдают звуковой сигнал с помощью динамика IBM/PC, каждый раз когда выдается код CHR$(7). Предположим, что Вам необходимо подавить выдачу звукового

сигнала. Для этого можно написать программу, которая будет проверять какие символы выдаются на экран и при появлении кода CHR$(7) заменяет его пробелом. После окончания проверки эта программа должна передавать управление обычной процедуре обслуживания дисплея.Фактически, нужно сделать небольшое добавление перед стандартной процедурой обслуживания дисплея. Чтобы задействовать эту программу необходимо заменить вектор прерывания дисплея (прерывание 16) адресом Вашей программы.

Прогаммные прерывания очень важны для работы IBM/PC. Большая часть этой книги будет посвящена их более подробному описанию. В главе 4 описываются прерывания системы BIOS в ПЗУ. Чтобы Вы могли извлечь максимальную пользу из этих системных функций, пакет программ, прилагаемый к данной книге, содержит полный набор интерфейсных процедур, как ассемблерных, позволяющих осуществлять доступ к служебным процедурам по прерыванию из любых языков программирования (включая Бейсик и Паскаль), так и вспомогательных процедур на Паскале, существенно облегчающих использование ассемблерных процедур.

Прежде чем завершить рассмотрение прерываний, необходимо упомянуть еще об одном необычном применении таблицы векторов прерываний. Таблица векторов прерываний, располагающаяся в самом начале оперативной памяти, предназначенная для хранения полных сегментированных адресов программ, которые должны обслуживать соответствующие прерывания. Однако, в IBM/PC использование таблицы векторов несколько шире. Хотя считается, что таблица должна хранить только адреса программ, разработчики IBM/PC сочли ее очень удобным местом для хранения адресов трех элементов, которые вовсе не являются программами. Это адреса очень важных системных данных. Идея заключается в том, чтобы использовать таблицу векторов прерываний в качестве унифицированного хранилища важных сегментированных адресов - в первую очередь, конечно, адресов программ, но, если необходимо, то и данных. Таким образом, три номера прерываний, 29, 30 и 31 не используются и соответствующие им вектора в таблице обнуляются, чтобы подготовить место для хранения адресов трех важных таблиц данных. Естественно, что эти номера прерываний не могут использоваться: если программа запросит прерывание с номерами 29, 30 или 31, то управление будет передано в одну из таблиц данных, которая начнет выполняться как программа.

В главе 6 мы более подробно рассмотрим ПЗУ и установим, что это за таблица. Затем мы рассмотрим их по одной, в главах 7, 8, 9.

3.6. СТЕК

Стеки - это наиболее интересные и полезные средства современных компьютеров. Вместе с прерываниями стеки обеспечивают эффективность работы компьютеров.

Что такое стек? Стек - это место, в котором компьютер хранит рабочие величины, так что один набор величин никогда не смешивается с другим набором.

Стек получил свое имя благодаря метафоре, которую часто используют для описания его работы. Представьте себе стопку тарелок, например, таких которые используются в кафетериях, где тарелки находятся на подпружиненной платформе, перемещающейся вверх и вниз. Если нужно добавить в стопку (по-английски stack) чистые тарелки, которые были в стопке, опускаются вниз. Когда кто-нибудь снимает тарелку сверху, вся стопка поднимается вверх. Тарелки используются по принципу "последним вошел - первым вышел".

Когда компьютер занят работой и поступает прерывание, необходимо место для того, что бы заполнить состояние компьютера до начала обработки прерывания. Если еще одно прерывание поступит в процессе обработки первого, то необходимо запомнить и эту информацию. Когда обслуживание второго прерывания завершится, комьютер должен вернуться к выполнению своей последней перед прерыванием работы, в данном случае к обслуживанию первого прерывания. Для обслуживания прерываний и для многих других операций, которые должны выполняться по принципу перехода к последней из отложенных операций, наиболее естественным средством для запоминания состояния компьютера будет стек.

Организация стека в IBM/PC основана на выделении под стек определенной области оперативной памяти и использовании специального регистра сегмента стека, SS, указывающего адреса этой области памяти. Вершина стека указывается содержимым специального регистра, называемого указателем стека SP. Стек компьютера, в отличие от подпружиненной стопки тарелок, не перемещается физически, а остается на месте. Зато изменяется адрес вершины стека, указываемый содержимым регистра SP.

Данные помещаются в стек компьютера операцией PUSH, извлекаются операцией POP.

Когда возникает прерывание, адрес текущей программы, хранящийся в регистрах CS и IP, помещается в стек, затем адрес процедуры обслуживания прерывания загружается в эти регистры и начинается выполнение данной процедуры. До ячейки, указываемой регистром SP (указателем стека), хранятся данные всей предыдущей работы, ожидающей своего возобновления. За

указателем стека находится свободное пространство стека, которое может использоваться процедурой обслуживания прерывания, если ей требуется дополнительная рабочая область. Если возникнет еще одно прерывание, новая процедура обслуживания найдет для себя свободную область в стеке несколько дальше.

Когда завершается очередная процедура, она освобождается. Сначало из стека извлекаются все рабочие значения, а затем извлекается адрес точки приостанова предыдущей процедуры, который загружается в регистры CS и IP. В процессе выполнения всех этих действий механизм работы со стеком автоматически поддерживает последовательность их выполнения. Стеки используются не только для обслуживания прерываний, но и при вызовах одних программ другими. Как при вызовах, так и при обслуживании прерываний принцип один: выполнявшаяся работа должна быть запомнена на время,так что бы можно было начать новую. Когда новая работа будет завершена, необходимо вернуться к выполнению старой в порядке, обратном тому в котором работа приостановлена.

При вызове подпрограмм часто возникает необходимость передавать им параметры, что можно сделать с помощью стека. В третьем приложении к этой книге описывается стыковка программ на Паскале с программами на ассемблере и там можно увидеть, как стек используется для передачи параметров.

Стеки настолько существенны для работы компьютеров, что можно только удивляться их относительно недавнему включению в конструкцию компьютеров. Серия компьютеров 360, которая являлась основной продукцией фирмы "IBM" начиная с 1960-тых годов, не реализовывала концепцию стека, что сильно отразилось на возможностях всей этой серии. Существует красивая легенда о том, что руководитель, исключивший стековую архитектуру из конструкции компьютеров фирмы "IBM", был в последствии "сослан" во внутрифирменный аналог Сибири.

Использование стеков может быть гораздо шире, чем описывалось выше. Микропроцессор 8088 использует стек только для хранения контекста приостановленных программ. Для этой же цели стек использует большинство компьютеров. Можно, однако, полность переориентировать способ работы компьютера с командами и данными, так что все будет основываться на использовании стека. Такая архитектура, ориентированная на использование стека, была разработана фирмой "Барроуз" и все, кто хочет понять на сколько радикально может отличаться компьютер с такой архитектурой, от обычных компьютеров, подобных IBM/PC, может обратиться к описаниям компьютеров фирмы "Барроуз".

Имеется одна подробность в использовании стека

микропроцессором 8088, о которой необходимо знать, если Вам потребуется воспользоваться содержимым стека или проверить рабочий стек. Стек продвигается от верхних адресов памти к нижним. Это означает , что старое содержимое стека имеет большие абсолютные адреса памяти, чем указатель стека SP. Так, например, чтобы извлечь параметры подпрограммы, помещенные в стек, используются положительные смещения (эт о описано в 3 приложении к этой книге). Что касается свободной части стека, к которой обычно обращаются только командами записи или извлечения из стека, а не по прямому адресу памяти, то для нее потребовалось бы отрицательное смещение относительно указания стека.

3.7. Порт

Под портом понимают совокупность средств унифицирующих и упрощающих способ взаимодействия микропроцессора 8088 с внешним миром. Порт обеспечивает единственную возможность приема или передачи данных любым обьектам кроме памяти.

Всем устройствам, с которыми должен взаимодействовать процессор, таким как клавиатура, дисководы или динамик, выделяется собственный порт. Порт представляет собой гипотетический путь данных, которому присвоен номер порта и который может принимать или передавать данные по команде процессора. Когда микропроцессор 8088 должен передавать данные в порт, используется команда OUT, которой указывается номер порта и передаваемые данные,имеющие длину один или два байта. Фактически команда OUT приказывает конкртному порту принять данные. Команда IN работает аналогично, только данные считываются из порта. Микропроцессор 8088 не может определить какие порты задействованы, а какие нет, так что все команды IN и OUT выполняются вслепую.

Операторы Бейсика обеспечивают прямой доступ к портам. Операторы Бейсика INP и OUT предоставляют те же возможности, что и машинные команды IN и OUT. Ниже, просто для примера, приведен фрагмент программы на Бейсике, работающей с динамиком:

```
10 X=INP(97) `чтение порта управления динамиком  X
20 REM       вероятно будет иметь значение 76 - про-
             верьте какое значенте Вы получите
30 OUT 97,X+3 `установить биты управления динамиком -
             прозвучит сигнал
40 OUT 97,X  `снять биты управления - окончание звука
```

Порты могут использоваться в сочетании с прерывниями. Например,если нажать клавишу на клавиатуре IBM/PC, никакие данные в компьютер не передаются. Вместо этого генерируется прерывание номер 9, указывающий, что имеются данные, которые нужно вводить с клавиатуры. В ответ на прерывание BIOS в ПЗУ выдаст команду IN для порта клавиатуры. Только после этого данные, определяющие какая клавиша была нажата, попадут в компьютер.

В случае с памятью может существовать до 1024К различных ячеек, причем компьютер не знает, какие именно адреса действительно имеются. То же самое касается и портов - микропроцессор 8088 может обращаться к порту с любым допустимым номером, не зная, работает ли этот порт или нет.

Адреса портов задаются 16 разрядами, так что потенциально можно иметь 64К различных портов. В действительности используется гораздо меньше номеров и, следовательно, имеются неограниченные возможности для расширения.

Имеется несколько способов использования портов в IBM/PC. Первый способ - это использование порта просто как пути данных. Например, коды клавиш, указывающие какая из клавиш была нажата, проходят через один порт; данные, выдаваемые на устройство печати, проходят через другой порт.

Еще один способ использования портов заключается в передаче через них управляющей информации для внешних устройств и получении от этих устройств информации о состоянии. Например, инициализация адаптера монохромного дисплея выполняется через порт 952. Еще одно назначение портов заключается в считывании положений переключателей в системном блоке, указывающих конфигурацию системы.

Ниже перечисляются некоторые наиболее важные применения портов. Как и некоторые подробности организации аппаратных средств, описанные в главе 2, номера конкретных портов могут не представлять интереса для читателя, но я привожу их полностью просто для полноты изложения.

Порт 96 (шестнадцатиричное 60) используется для передачи данных в формате порядковых номеров, от клавиатуры. При описании клавиатуры в главе 10 мы рассмотрим форматы номеров и их использование более подробно.

Порт 97 (61(16)) используется для управления встроенным динамиком, а также двигателем кассетного механизма. Он также позволяет запустить аппаратный таймер, что более подробно мы рассмотрим в главе 11.

Порты с 64 по 67 (40(16)-43(16)) используются для управления программируемым таймером, используемым как

динамиком, так и интерфейсом кассетного накопителя. О том, как используются все эти порты, будет рассказано в главе 11.

Монохромный дисплей использует несколько последовательных портов, начиная с порта 944(3B0(16)), а цветной графический - последовательность, начинающуюся с порта 976(3D0(16)). Контролер гибких дисков использует последовательность портов, начиная с порта 1008(3F0(16)), а собственно данные, записываемые на дискету или считываемые с нее, передаются через порт 1013(3F5(16)).

Вместе с программой 3.1 для исследования конфигурации памяти, листинги 3.2 и 3.3 показывают программу, объединяющую модули, написанные на Паскале и на ассемблере, которая считывает все порты и сообщает о том, какие из них наиболее вероятно активны. Готовая к запуску версия этой программы имеется на дискете, прилагающейся к этой книге. В отличие от программы обследования памяти эта программа не дает достаточно практичного результата, однако, она может оказаться довольно интересной.

Поскольку Бейсик обеспечивает доступ к портам, эту программу можно было бы написать на Бейсике, но тогда для читателя не было бы в ней ничего нового. Вместо этого приведенный пример может послужить хорошим введением в программирование на ассемблере, Паскале и примером объединения программ, написанных на разных языках.

Приложение 3.1. Текст программы поиска активного участка памяти (Бейсик).

```
1000 REM
1010 REM
1020 REM
1030 GOSUB 2000 ' TITLE
1040 GOSUB 3000 ' SEARCH AND DISPLAY
1050 GOSUB 4000 ' RETURN TO DOS

2000 REM
2010 KEY OFF : CLS : WIDTH 80
2020 REM
2030 PRINT "
2040 PRINT "
2050 PRINT
2060 PRINT "
2070 PRINT
2080 PRINT "
```

```
    2090 PRINT ''
    2100 PRINT
    2999 RETURN

    3000 REM
    3010 TRUE.% = -1
    3020 FALSE.% = 0
    3030 IN.MEMORY.% = FALSE.%
    3040 FOR PARAGRAPH.! = 0 TO 65535! STEP  64  '
CHECK EACH 1K OF MEMORY
    3050 GOSUB 5000 ' CHECK FOR ACTIVE MEMORY
    3060     IF     (IN.MEMORY.%=FALSE.%)     AND
(MEMORY.HERE.%=TRUE.%) THEN GOSUB 6000
    3070     IF     (IN.MEMORY.%=TRUE.%)     AND
(MEMORY.HERE.%=FALSE.%) THEN GOSUB 7000
    3080 IN.MEMORY.% = FALSE.%
    3090 IF MEMORY.HERE.% THEN IN.MEMORY.HERE.% =TRUE.%
    3100 NEXT PARAGRAPH.!
    3110  IF  IN.MEMORY.%  THEN  PARAGRAPH.!  =  65536  :  GOSUB
7000
    3999 RETURN

    4000 REM
    4010 PRINT
    4020 PRINT "Finished."
    4999 SYSTEM

    5000 REM
    5010 DEF SEG = PARAGRAPH.!
    5020 BYTE0.% = PEEK (0)
    5030 BYTE1.% = PEEK (1)
    5040 BYTE2.% = PEEK (2)
    5050 BYTE3.% = PEEK (3)
    5060 CHECK.COUNT.% = 0
    5070    IF    BYTE0.%    =    233    THEN    CHECK.COUNT.%    =
CHECK.COUNT.% + 1
    5080    IF    BYTE1.%    =    233    THEN    CHECK.COUNT.%    =
CHECK.COUNT.% + 1
    5090    IF    BYTE2.%    =    233    THEN    CHECK.COUNT.%    =
CHECK.COUNT.% + 1
    5100    IF    BYTE3.%    =    233    THEN    CHECK.COUNT.%    =
CHECK.COUNT.% + 1
    5110 MEMORY.HERE.% = FALSE.%
    5120  IF  CHECK.COUNT.%  <=  3  THEN  MEMORY.HERE.%  =
TRUE.%
    5999 RETURN
```

```
    6000 REM
    6010 IN.MEMORY.% = TRUE.%
    6020 START.! = PARAGRAPH.!
    6999 RETURN

    7000 REM
    7010 SIZE.! = (PARAGRAPH.!-START.!) * 16
    7020 IF SIZE.! < 8 * 1024 THEN 7999 '
        SUPPRESS SMALL-BLOCK FALSE REPORTS
    7030 PRINT "Active memory begins at ",
    7040 PRINT HEX$(START.!)
    7050 PRINT "                ends at ",
    7060 PRINT HEX$(PARAGRAPH.!-1);
    7070 PRINT SIZE.! / 1024;"K-bytes"; ("SIZE.!;"bytes )"
    7080 IN.MEMORY.% = FALSE.%
    7090 PRINT
    7999 RETURN

    9999 REM
```

Приложение 3.2. Текст программы обработки прерываний (Ассемблер).

```
beepseg segment 'code'


    assume  cs:beepseg

beep    proc   far

        cli

        mov   bx,3000
        in    al,61h
        push  ax
more:   and   al,0fch
        out   61h,al
        mov   cx,50
11:     loop  11
        or    al,2
        out   61h,al
        mov   cx,50
12:     loop  12
        dec   bx
```

```
        jnz   more
        pop   ax
        out   61h,al

        sti

        int   20h
beep    endp

beepseg  ends

end
```

Приложение 3.3. Текст программы поиска активных портов (Паскаль).

```
program porttest (output);

function inport (x : word) : byte;
  external;

var
  count : word;
  b     : byte;
  w     : word;
  c     : array [wrd(0)..255] of word;
  headc : word;

procedure header1;
 var [static]
   i : integer;
 begin
   writeln;
   for i := 1 to 8 do
     write (' Port Val');
   writeln;
 end;

procedure header2;
 var [static]
   i : integer;
 begin
   writeln;
   for i := 1 to 8 do
     write (' Val Count');
```

```
  writeln;
 end;
procedure initialise;
  begin
    count := 0;
    headc := 0;
    for b := 0 to 255 do
      c[b] := 0;
    for w := 1 to 25 do
      writeln;
    writeln('Program for INSIDE THE IBM PERSONAL COMPUTER');
    writeln('(C) Copyright Peter Norton, 1983');
    writeln('Listing 3.3: PORTTEST - read all ports');
    writeln;
    writeln('The following may be active ports:');
    header1;
  end;

procedure scan_all_ports;
  begin
    for w:=0 to maxword do
      begin
        b:=inport(w);
        c[b]:=c[b]+1;
        if not(b in [wrd(0),78,110,188,202,203,207,254,255])
        then
          begin
            write(w:6,b:4);
            count:=count+1;
            headc:=headc+1;
            if headc>159 then
              begin
                headc:=0;
                header1;
              end;
          end;
      end;
    header1;
  end;

procedure finish_up;
  begin
    writeln;
    writeln('Here is a profile of the values returned for ',
            all of the possible ports:');
    header2;
```

```
  for b:=0 to 255 do
    write(b:4,c[b]:6);
  header2;
  writeln(count,' ports may possibly be active');
  writeln;
  writeln('Finished.')
end;
begin
  initialize;
  scan_all_ports;
  finish_up
end.
```

Приложение 3.4. Текст программы считывания данных из порта (Ассемблер).

```
inport_code segment 'code'

public  inport

inport  proc  far
        push  bp
        mov   bp,sp
        mov   dx,[bp+6]
        in    al,dx
        pop   bp
        ret   2

       db    '(C) Copyright Peter Norton, 1983'

inport  endp

inport_code ends

      end
```

ГЛАВА 4. ОСНОВЫ ОРГАНИЗАЦИИ ДОС

[Оглавление]

В этой главе будет кратко рассмотрена система ДОС, основная операционная система IBM/PC. Мы не будем слишком

подробно рассматривать ее, частично потому что ДОС может составить достаточный предмет для нескольких книг сама по себе, а частично потому, что эта книга в основном посвящена внутренней организации IBM/PC, а не организации ДОС.

Но все же, поскольку ДОС очень широко используется на IBM/PC, необходимо иметь определенные представления об ее организации. Так как основное внимание в этой книге уделяется наиболее сложным возможностям IBM/PC, мы покажем как осуществляется доступ к этим возможностям с помощью средств операционной системы ДОС.

Здесь тесно соприкасаются все три упоминавшиеся выше области интересов. Практически все, о чем пойдет речь в данной главе, относится к IBM/PC, совместимым с ней компьютерам, а также к компьютерам, работающим под управлением MS-DOS.

4.1. Для чего нужны операционные системы?

Чтобы полнлстью овладеть всеми возможностями своего компьютера, необходимо понимать его операционную систему. Предлагаемый в этой главе краткий обзор позволит Вам понять, что собой представляет ДОС. Слишком подробных знаний Вы не приобретете, но ознакомление с этой главой позволит Вам начать с ней работать.

Назначение операционной системы заключается в обеспечении удобства управления компьютером. Операционная система, в полном смысле этого термина, является первой и наиболее важной программой любого компьютера. Как правило, она является и наиболее сложной. Паразительно, что наиболее совершенные программы используются только для управления самим компьютером. Ирония здесь заключается в том, что компьютеры и программы для них разрабатывались для выполнения полезной работы, а не как самоцель. Но с другой стороны, наиболее мощный инструмент, созданный человеком, компьютер, имеет достаточно возможностей, чтобы работать сам по себе и человек не смог бы управлять им, если бы не операционная система.

Большая часть работы операционной системы заключается в том, чтобы скрыть от пользователей некоторые сложные и ненужные им детали. В качестве иллюстрации рассмотрим работу команды COPY в ДОС. Предположим, что мы используем ее для копирования с одной дискеты на другую. Вам может показаться, что это не очень сложная процедура. Рассмотрим, однако, лишь некоторые действия, которые должна выполнить операционная система:

- Проверить, имеется ли на исходной дискете файл с указанным именем.

- Проверить, должно ли копирование выполняться не в файл на дискете, а на какое-нибудь устройство (например, на устройство печати).
- Проверить, нет ли файла с таким именем на целевой дискете.
- Проверить, достаточно ли места на целевой дискете для размещения файла (учитывая все свободное место на дискете, а, если копия файла уже есть на дискете, то еще и место, которое она занимает).
- Если на целевой дискете должен создаваться новый файл, то необходимо проверить, имеется ли свободное место в справочнике или он уже заполнен.
- Установить формат исходной дискеты: односторонний или двухсторонний?
- Установить формат целевой дискеты: односторонний или двухсторонний?
- Проверить, не копируется ли файл сам в себя (что запрещено).
- Проверить наличие в системе двух дисководов, поскольку для систем с одним дисководом необходимо имитировать диск B.
- Проверить, необходимо ли вычислять размер исходного файла по его размеру, указанному в справочнике, или он будет определяться маркером конца файла (как для текстовых файлов в коде ASCII).
- Проверить, находится ли таблица размещения файлов для исходной дискеты в оперативной памяти.
- Проверить, находится ли таблица размещения файлов для целевой дискеты в оперативной памяти.
- Проверить корректность размещения исходного файла на дискете по соответствующей таблице размещения файлов.
- Проверить, совпадает ли размер файла, указаный в справочнике, с действительным размером.
- Установить, какой обьем памяти можно использовать в качестве буфера при перезаписи файла.
- Проверить, больше или меньше 64К размер буфера.
- Нужно ли удалять из памяти интерпретатор команд, чтобы увеличить размер буфера?

Вы еще не устали от этого перечисления? А ведь это еще только начало! Пока что рассмотрен только логический уровень задачи. Ниже приводится физический уровень, причем при его изменении опущено значительно больше подробностей:

- Запущен ли двигатель дисковода?
- Выведена ли головка чтения/записи на нужную дорожку?
- Сколько секторов данных нужно считать/записать на этой дорожке?

- Готов ли дисковод к выполнению команды?
- Работает ли дискета? Не требуется ли перезапуск, повторная попытка выполнения операции или повторное позиционирование головки?
- Если уже выполнялась повторная попытка, достаточно ли было сделано попыток, чтобы выдать запрос о дальнейших действиях?
- Каким был ответ на запрос? Повторить, игнорировать или аварийно завершить работу?
- Ожидание готовности дисковода?
- Операция чтения/записи/поиска завершилась успешно?
- Целевая дискета защищена от записи?

Все описанное выше является лишь приблизительным перечислением подробностей операции копирования файла. А это еще относительно простая операция ДОС. Кстати, приведенное разделение на логический и физический уровни очень важно для работы операционной системы.

Основная забота операционной системы, такой как ДОС, заключается в том , чтобы скрыть от пользователя эти ненужные ему подробности. Значительная часть функций операционной системы заключается в обслуживании устройств ввода/вывода.

Помимо физического уровня обслуживания ДОС обеспечивает и услуги более высокого уровня, такие как поиск в справочниках, копирование файлов и загрузку программ.

Один из путей достижения эффективности операционных систем - это модульность их разработки. Операционная система упрощается и становится более эффективной, если ее разработчики разделят все ее функции на отдельные части. Затем эти части должны быть организованы в тщательно спланированную иерархическую систему, в которой каждый уровень иерархии выполняет присущие ему функции, освобождая уровни, находящиеся выше по иерархии, от их выполнения (и, в свою очередь, не занимаясь деталями функций, присущих нижним уровням). Теперь мы перейдем к рассмотрению отдельных частей ДОС.

4.2. Шесть основных модулей ДОС

Всю систему можно разделить на шесть основных частей, функции каждой мз которых мы кратко рассмотрим в этом разделе. Первая часть - это система BIOS в ПЗУ. Это система поставляется вместе с IBM/PC и может использоваться любой операционной системой. Функция BIOS в ПЗУ заключается в обслуживании основных и наиболее фундаментальных операций

компьютера. Поскольку система BIOS в ПЗУ для IBM/PC является "встроенной" в комптютер, она является не только составной частью ДОС, но и составной частью любой операционной системы IBM/PC.

Еще одна часть - "загрузочная запись" дискеты. Загрузочная запись - это очень короткая и простая программа, находящаяся в первом секторе каждой дискеты. Функция этой программы заключается в запуске процесса загрузки операционной системы после включения компьютера или перезапуска (нажатия клавиш CTRL-Alt-Del). Загрузочная запись считывает еще два модуля операционной системы в память и они завершают процесс загрузки ДОС.

Следующие две части ДОС представляют собой дисковые файлы IBMBIO.COM и IBMDOS.COM. Обе эти части загружаются в память загрузочной записью и остаются в памяти во время работы ДОС, так что провести между ними различие довольно трудно. Файл IBMBIO.COM представляет собой изменяемое дополнение к системе BIOS в ПЗУ. Начиная с версии 2.00 файл IBMBIO.COM может дополняться другими частями, называемыми независимыми драйверами устройств, которые мы подробнее рассмотрим несколько позже. Система BIOS в ПЗУ, IBMBIO.COM и драйверы устройств, вместе взятые образуют "физический" уровень операционной системы.

Файл IBMDOS.COM реализует основные услуги ДОС. Это "логический" уровень ввода/вывода операционной системы.

Оба эти файла являются "скрытыми" системными файлами (о которых подробнее будет сказано в главе 5). Имена этих файлов не включаются в оглавление диска, вызываемое по команде DIR. Команда CHKDSK (для ДОС версии 1.10 и последующих) укажет, что эти файлы находятся на дискете, но не сообщит их имена. Если Вы воспользуетесь процедурой DiskLook для просмотра содержимого дискеты, то оба имени, IBMBIO.COM и IBMDOS.COM, будут присутствовать в списке и их можно просматривать так же, как все остальные файлы. Одна из функций IBMBIO.COM заключается в загрузке еще одного модуля операционой системы.

Пятая часть - это дисковый файл COMMAND.COM. Главная функция COMMAND.COM заключается в обработке команд, вводимых
пользователем. Команды ДОС, считающиеся внутренними, такие как TYPE, COPY и DIR фактически реализуются программами COMMAND.COM. Собственно COMMAND.COM разделяется на две части:
одна становится частью IBMDOS.COM, а вторая - процессором дополнительных команд. Более подробно об этом сказано в разделе 4.7.

Шестая и последняя часть ДОС состоит из всех внешних

команд, таких как FORMAT и DISKCOPY. Функции этих команд различны и реализующие их программы загружаются в память только в случае необходимости. В отличие от остальных пяти частей ДОС, все эти дополнительнве программы не являются интегральной составляющей ДОС, хотя они и поставляются вместе с ней. Некоторые из этих дополнительных программ, например, FORMAT, настолько важны, что обойтись без них практически невозможно.

Внешние команды являются нерезидентными частями ДОС, поскольку они не находятся в памяти постоянно. Файлы IBMBIO.COM, IBMDOS.COM и драйверы устойств находятся в памяти
постоянно и образуют резидентную часть ДОС. Файл COMMAND.COM
можно выделить в отдельную категорию как полу-резидентную составляющую ДОС, о чем речь пойдет ниже. Загрузочная запись используется только временно и поэтому не является резидентной частью ДОС. В следующих шести разделах мы более подробно рассмотрим каждую из частей ДОС.

4.3. Система BIOS в ПЗУ

Первая часть в ДОС - это BIOS в ПЗУ или базовая система ввода/вывода, размещающаяся в постоянной памяти. Эта система обеспечивает наиболее простые и универсальные услуги операционной системы, связанные с осуществлением ввода/вывода.

Система BIOS располагается в ПЗУ начиная с адреса FE00 и до FFFF, сразу же после интепретатора БЕЙСИКа. Распределение памяти, приведенное в главе 3, показывает и размещение системы BIOS.

Поскольку BIOS в ПЗУ является частью IBM/PC, она может изменяться только в том случае, когда изменениям подвергается аппаратная реализация системы. В качестве составной части IBM/PC, система BIOS в ПЗУ является не просто одной из частей ДОС, но частью любой операционной системы IBM/PC.

Система BIOS в ПЗУ состоит из нескольких частей, большинство из которых представляет собой программы (остальные - это важные таблицы данных, которые мы рассмотрим в последующих главах). Программа системы BIOS, которая выполняется первой, представляет собой тест функциониования.

Эта программа поверяет память и внешние устройства, подключенные к IBM/PC, как только будет включено питание компьютера. Работа этой программы определяет ту задержку, которая имеет место между включением питания и загрузкой

операционной системы. Чем больший объем памяти подключен к Вашему компьютеру, тем дольше выполняется тест, поскольку наиболее длительной процедурой является процедура поверки памяти.

Следующая часть BIOS, которая должна выполняться как программа запуска операционной системы, - это программа вызова загрузчика операционной системы. Эта программа проверяет, подключен ли дисковод с гибкими дисками и считывает с дискеты "загрузочную запись". После считывания загрузочной записи программа запуска передает ей управление, чтобы она считала оставшиеся части операционной системы.

Если в системе нет дисковода или при считывании загрузочной записи произошла ошибка, то программа запуска BIOS передает управление кассетной системе БЕЙСИК. Если Вы включите IBM/PC, не установив дискету в дисковод, то Вы увидите, что начала работу программа кассетной системы БЕЙСИКа.

Помимо двух уже описанных частей - автономного теста функционирования и программы запуска, система BIOS в ПЗУ содержит множество других программ и они-то представляют для нас наибольший интерес. Сюда входят программы обслуживания всего стандартного периферийного оборудования IBM/PC. Эти программы выполняют основные функции управления клавиатурой, дисплеем, дискетами, асинхронным адаптером связи, устройством печати и кассетным интерфейсом. Работой с этими программами нам необходимо овладеть, чтобы полностью использовать все возможности IBM/PC. Подробно мы рассмотрим их в главах с 6 по 11.

4.4. Начальная загрузка - загрузочная запись

Загрузочная запись необходима для запуска ДОС. Принцип использования загрузочной записи реализован практически на всех компьютерах. Загрузочная запись содержит минимум необходимых команд для считывания и запуска основных частей операционной системы.

При запуске IBM/PC либо включением питания, либо нажатием клавиш Ctrl-Alt-Del, процедура запуска системы BIOS в ПЗУ считывает первую запись дискеты, установленной в дисковод A и помещает ее в стандартную область памяти, по адресу 31744 или 7C00 (шестнадцатиричное). После считывания загрузочной записи BIOS передает ей управление, выполняя переход по адресу 31744. Далее загрузочная запись должна продолжить загрузку операционной системы.

Основное назначение загрузочной записи для ДОС заключается просто в загрузке файлов IBMBIO.COM и IBMDOS.COM.
Загрузочная запись имет размер стандартного сектора на дискете, 512 байт, что недостаточно для сложной программы. Для упрощения работы этой программы, оба файла, IBMBIO.COM и IBMDOS.COM, размещаются в определенных постоянных местах на
дискете. Это избавляет программу загрузки от необходимости искать их в справочнике дискеты.

Этим, кстати, "системная" дискета отличается от обычной - на ней, в определенных местах, находятся два системных файла IBMDOS.COM и IBMBIO.COM. Поэтому нельзя просто преобразовать обычную дискету в системную - места, зарезервированные для системных файлов, могут быть заняты другими файлами.

Хотя загрузочная программа не столь совершенна, чтобы осуществлять поиск системных файлов, она все же способна проверить правильность их включения в справочник. Поскольку оба системных файла включены в справочник дискеты, они защищены как от стирания, так и от любых других способов доступа установлением атрибутов скрытого и системного файла. (В следующей главе мы поясним, что такое скрытые и системные файлы).

В связи с простотой ее задачи, программа загрузки является относительно стабильной составной частью ДОС. Ее потребовалось изменить, когда изменялся размер или местоположение системных файлов, а это произошло с введением поддержки двухсторонних дискет в версии ДОС 1.10.

Вы можете обнаружить различия в загрузочных записях ДОС различных версий. (Загрузочную запись легко просмотреть с помощью моей утилиты DiskLook: запустите DiskLook, нажмите клавишу F7 и введите адрес загрузочной записи на дискете: сторона 0, дорожка 00 и сектор 1, после чего нажмите клавишу F6 - на экране будет отображена загрузочная запись).

Загрузочная запись ДОС версии 1.00 имеет одно незначительное отличие от всех остальных версий, которое легко заметить.Вместо имени Роберта О'Рира в исходной версии ДОС, во всех последующих версиях указывается название фирмы "Майкрософт".

4.5. Дополнительные операции с устройствами: IBMBIO.COM

Первый из двух системных файлов, IBMBIO.COM,

предназначен для дополнения функций системы BIOS в ПЗУ. Как BIOS в ПЗУ, так и IBMBIO.COM занимаются обслуживанием операций ввода/вывода или обслуживанием устройств, что то же самое. Такая функция предполагает обработку различных подробностей, связанных с функционированием устройств ввода/вывода. Сюда же включается обнаружение ошибок и их исправление, что для программ еще сложнее.

Программы IBMBIO.COM отличаются тем, что их легко можно изменить, чего нельзя сказать о программах BIOS в ПЗУ. Файл IBMBIO.COM предназначен для решения трех задач, которые невозможно решить с помощью системы BIOS в ПЗУ. Первая задача
заключается в настройке на нужды конкретной операционной системы, ДОС. Любая операционная система, включая CP/M-86 и р-систему Калифорнийского университета в Сан-Диего, может использовать универсальную систему BIOS в ПЗУ, но в той части, в которой системы различаются, они должны реализовывать собственные фрагменты системы ввода/вывода.

Вторая задача IBMBIO.COM заключается в исправлении любых ошибок в BIOS в ПЗУ, если в этом возникнет необходимость. Программы, подобные системе BIOS, проверяются очень тщательно, ведь их размещение в ПЗУ не допускает исправлений. Если ошибки все же будут позднее обнаружены в BIOS, их можно будет исправить путем внесения соответствующих изменений в IBMBIO.COM. Это делается путем изменения векторов прерываний таким образом, чтобы управление при обращении к операциям BIOS сначала попадало в IBMBIO.COM, а уже затем в BIOS. Соглашения о вызовах процедур системы BIOS в ПЗУ всегда предполагает использование прерываний, вместо прямых переходов по адресам ПЗУ. Основная причина использования прерываний как раз и состоит в обеспечении возможностей перехвата управления у процедур BIOS, ведь, в противном случае, такой возможности просто не существовало бы. В главе 7, когда будет рассматриваться программа управления дискетами, известная под названием "Disk base", Вы увидите одно из таких изменений, частично подменяющее функции системы BIOS в ПЗУ.

Третья задача, которую не позволяет решить BIOS, и, которую за нее решает IBMBIO.COM, заключается в обслуживании новых периферийных устройств, таких как жесткие диски большой емкости, или восьмидюймовые (203мм) дискеты, или плоттеры, или любые из сотен устройств, которые могут быть подключены к IBM/PC. Когда новое устройство ввода/вывода подключается к IBM/PC, обслуживающая его программа должна включаться в файл IBMBIO.COM или его расширения, без необходимости замены микросхем ПЗУ, в которых размещается система BIOS.

В наиболее ранних версиях ДОС задача включения нового устройства в систему предполагала внесение изменений в IBMBIO.COM и, возможно, в другие системные программы ДОС. Хотя это не слишком сложно для таких фирм как "ИБМ" или "Майкрософт", такая задача может оказаться непосильной для пользователя, которому требуется включить дополнительное оборудование в свою систему.

Поскольку возможность добавления новых устройств к компьютеру составляет немаловажное условие его успеха на рынке, процедура включения обслуживающих программ в ДОС была
упрощена, начиная с ДОС версии 2.00. Когда управление впервые после запуска системы передается IBMBIO.COM, он проверяет, находится ли на дискете файл конфигурации системы. Если файл конфигурации найден, считываются его команды, часть которых определяет ряд параметров системы.

Команды файла конфигурации включают имена всех программ обслуживания устройств, которые необходимо включить в BIOS. Каждая из этих программ загружается в память как дополнение к IBMBIO.COM. Такая схема облегчает добавление новых устройств, позволяет делать это модульно, не затрагивая системные файлы ДОС.

Как правило, любая программа, работающая на IBM/PC, будет использовать обычную версию BIOS для ДОС. Однако, в некоторых случаях программе необходимо бывает особое обслуживание операций ввода/вывода. Поскольку IBMBIO.COM представляет собой дисковый файл, который можно изменить, имеет смысл разработать версию IBMBIO.COM, настроенную на Ваши конкретные нужды. Пример такой настройки Вы можете найти в диагностических программах.

4.6. Основа ДОС: IBMDOS.COM

Разделение операционной системы на два модуля, IBMBIO.COM и IBMDOS.COM позволяет разделить те части, которые
специфичны для конкретного компьютера и те, которые являются общими для всех компьютеров, работающих под управлением ДОС.

Служебные процедуры ДОС, в определенной степени произвольно, разделены на те, которые вызываются с помощью собственных прерываний, и те, которые разделяют общее прерывание (номер 33, шестнадцатиричное значение 21). В терминологии ДОС, первая группа называется прерываниями ДОС, а

вторая - вызовами функций. В обоих случаях для их вызова используются программные прерывания, причем по той же причине, что и для вызова программ BIOS: для обеспечения модульности.

Диапазон прерываний от 32 до 63 (шестнадцатиричные значения от 20 до 3F) зарезервированы для использования ДОС. Используется только часть этих прерываний, а остальные обеспечивают возможность дальнейшего расширения. Программы обслуживания прерываний ДОС включают чтение и запись секторов дискеты, доступ к контролю за ошибками ДОС и операциями клавиатуры Ctrl-Break.

Вызовы функций ДОС обеспечивают, в основном, промежуточный уровень обслуживания операций ввода/вывода. В качестве примеров можно упомянуть ввод с клавиатуры, обычный вывод на экран дисплея, ввод/вывод по асинхронной линии связи и вывод на печать. Предусмотрены также логические операции для дискет - открытие и закрытие файлов, поиск в справочнике файлов, удаление и создание файлов, чтение и запись данных. Эти процедуры обеспечивают практически все элементарные операции, которые могут потребоваться программе для работы с файлами и данными, хранящимися в файле, так чтобы программе не приходилось самостоятельно расшифровывать данные справочников, таблиц размещения файлов и т.п.

Большая часть этих служебных процедур ДОС интенсивно используется программами более высокого уровня ДОС. Например, процедура поиска в справочнике используется командами DIR и COPY. Кроме того, она используется интерпретатором команд для поиска программных файлов. В разделе 4.9 будет рассмотрен полный список служебных процедур ДОС.

4.7. Файл COMMAND.COM и внутренние команды

Следующая составная часть ДОС, файл COMMAND.COM, одна из наиболее интересных частей, как с точки зрения тех услуг, которые она предоставляет пользователю, так и с точки зрения принципов ее функционирования. COMMAND.COM имеет несколько функций. Прежде всего, это "процессор команд", что означает возложенные на него функции ввода команды, набираемой на клавиатуре, и определения дальнейших действий.

Когда вводится внутренняя команда, такая как DIR, COPY, TYPE, REM или PAUSE, то запрашивается случайная процедура, "встроенная" в COMMAND.COM, так что она может выполняться немедленно.

Чтобы распознавать внутренние команды, COMMAND.COM содержит таблицу имен команд. Если просмотреть файл COMMAND.COM, то можно увидеть эти имена команд. Для просмотра
файла COMMAND.COM можно воспользоваться программой DEBUG или
DiskLook. Там же Вы увидите сообщение, которое ДОС выдает в начале работы. При желании эти сообщения можно изменить с помощью команды DEBUG или SecMod, так чтобы ДОС начинала работу выдачей Вашей фамилии или названия компании. Можно также изменить и имена внутренних команд, причем сделать это просто, если не изменяется длина имени.

Если команды нет в таблице внутренних команд, значит имеется в виду внешняя команда, которую COMMAND.COM будет искать во внешнем файле. В ответ на наш запрос, COMMAND.COM осуществляет поиск файла обработки команды на соответствующей дискете и запускает его выполнение.

Существует три типа файлов обработки команд, поиск которых в определенном порядке осуществляет COMMAND.COM. Название команды точно совпадает с именем файла, в котором хранится програма обработки команды. Три типа файлов обработки команд различаются расширениями имен файлов. Три расширения, в порядке их приоритетности: ".COM", которое обозначает программный файл в одном из двух программных форматов; ".EXE", обозначающее программный файл другого формата и ".BAT", обозначающее файл пакетной обработки. (Форматы этих файлов и масса другой информации, касающейся файлов на дискетах, приведены в следующей главе.)

Когда COMMAND.COM находит программный файл в любом формате, он выполняет загрузку этого файла в память и выполняет любое необходимое преобразование. После загрузки файла и формирования приставки программного сегмента, COMMAND.COM передает управление этой программе, так чтобы она
могла выполнить свою работу.

Если файл обработки команды относится к типу ".BAT", то есть является файлом пакетной обработки, то он содержит в формате текстового файла в коде ASCII последовательность команд, которые должны выполняться так, как будто они вводятся с клавиатуры. Одна из многих задач COMMAND.COM заключается в отслеживании позиции в файле пакетной обработки, так чтобы после выполнения одной команды сразу переходить к другой. Если в процессе обработки одного пакетного файла будет вызван другой, то возврата к первому не произойдет, так как файлы пакетной обработки не могут быть вложенными, зато их можно объединять в цепочку.

В некоторых системах весь ввод команд может быть переназначен на файл, причем это относится как к программам, вводящим данные с клавиатуры, так и к интерпретатору команд. К ДОС это не относится. Только интерпретатор команд COMMAND.COM может автоматически выполнять чтение из файлов
пакетной обработки.

Кстати, программы могут вносить изменения в файл пакетной обработки и таким образом определять, какая команда будет выполнена следующей. Это часто делается в сложных приложениях для замены последовательного выполнения программ.

Файл COMMAND.COM имеет значительно больше функций, чем было описано. Фактически, COMMAND.COM разделяется на три части. Первая часть размещается в памяти сразу после програм IBMBIO.COM и IBMDOS.COM и, подобно им, становится резидентной частью ДОС. Фактически эта часть COMMAND.COM не отличается от IBMDOS.COM.

Вторая часть COMMAND.COM используется только временно: при запуске системы она осуществляет поиск файла пакетной обработки AUTOEXEC.BAT и, если он найден, его выполнение. После выполнения этой функции данная часть COMMAND.COM уже не нужна.

Третья, наиболее интересная часть COMMAND.COM является полурезидентной и одной из наиболее хорошо проработанных частей ДОС. Эта часть включает интерпретатор команд и программы, реализующие внутренние команды ДОС. Необходимо
отдавать себе отчет в том, что такая сложная программа как интерпретатор команд не может быть очень компактной. С одной стороны, лучше всего было бы, чтобы интерпретатор команд постоянно находился в памяти. С другой стороны, не хотелось бы, чтобы он занимал место в памяти все время, особенно, если места не слишком много (в системах с 64К, например).

Интересное решение этой проблемы заключается в том, чтобы разместить эту часть COMMAND.COM в верхних адресах памяти (обычно используются последние адреса) и позволить другим программам затирать эту область памяти.

Когда снова возникает необходимость использования интерпретатора команд, резидентная часть COMMAND.COM сначала
проверяет находится ли интерпретатор команд в памяти. Если он запорчен другими программами, то резидентная часть перезагружает его с дискеты. (По этой причине, кстати, необходимо иметь копии COMMAND.COM на всех Ваших дискетах,
даже на тех, которые не форматировались в системном формате.

Если в процессе работы Вы увидите сообщение:"Insert DOS disk..." [вставьте диск ДОС], это будет означать, что на используемой Вами дискете отсутствует файл COMMAND.COM).

Чтобы проверить наличие в памяти интерпретатора команд, вычисляется контрольная сумма тех ячеек памяти, в которых он должен находиться. Если сумма не совпадает с ожидаемой, то COMMAND.COM перезагружается. Вычисление контрольной суммы
может производиться при перезагрузке COMMAND.COM и, если обнаружено различие, ДОС выдает сообщение об ошибке. Это происходит даже в том случае, когда различие заключается всего лишь в том, что Вы изменили начальное сообщение ДОС ("The IBM Personal Computer DOS"..), которое, как я уже упоминал, можно изменить. Если Вы это сделали или внесли какие-нибудь другие изменения в COMMAND.COM, это необходимо сделать на всех Ваших дискетах.

Одна из причин, по которой COMMAND.COM оформляется в виде отдельного файла и не объединяется с двумя другими системными файлами, заключается в облегчении разработки прикладных версий. Это одна из основных возможностей адаптации IBM/PC к нуждам пользователей. Если необходимо часть пользовательских команд сделать внутренними или изменить способ работы интерпретатора команд, то можно написать специальную программу COMMAND.COM.

Примером прикладной версии COMMAND.COM может служить исходная версия 1.00 текстового процессора Easy-Writer. Он имеет собственный вариант COMMAND.COM и поэтому переход от
ДОС к Easy-Writer и наоборот требует перезагрузки системы.

Как рекомендует руководство фирмы "ИБМ", операционная система может быть перезагружена после нажатия клавиш Ctrl-Alt-Del. Это приводит к полному перезапуску системы, начиная с чтения загрузочной записи. Имеется, однако, менее радикальный способ перезапуска. Если ввести команду COMMAND, то перезагрузится только COMMAND.COM и система перезапустится
без перезагрузки IBMBIO.COM и IBMDOS.COM. Это позволяет обновить версию интерпретатора команд и выполнить файл пакетной обработки AUTOEXEC.BAT.

4.8. Оставшиеся части ДОС: внешние команды

Последнюю часть операционной системы ДОС составляют внешние команды. Они называются внешними, поскольку они не

включены в состав ДОС, и не являются резидентными в памяти IBM/PC. Вместо этого, внешние команды хранятся в программных файлах на дискетах.

Как все программные файлы, все внешние команды имеют расширение имени файла ".COM" или ".EXE", что указывает, в каком из форматов они оформлены. Мы рассмотрим эти форматы, когда будем рассматривать структуру дискет и файлов в следующей главе.

Примерами внешних команд могут служить DISKCOPY, COMP, FORMAT, а такие программы как EDLIN, BASIC или BASICA тоже в определенном смысле являются внешними командами, хотя их лучше все же считать языковыми процессорами, такими как макроассемблер или Паскаль.

Внешние команды не отличаются от других программных файлов, которые Вы сами разрабатываете или покупаете. С одной точки зрения внешние команды действительно являются частью операционной системы, особенно те, без которых операционной системой нельзя воспользоваться, такие как FORMAT. Однако, с другой точки зрения, внешние команды - это всего лишь вспомогательные программы, утилиты, которые полезны, но которые не являются операционной системой. Разница между внешней командой ДОС и обычной программой заключается просто в Вашем отношении к ней - как к части ДОС или как к чему-нибудь малосущественному.

4.9. Доступ к средствам и услугам ДОС

В это разделе мы рассмотрим все служебные процедуры, доступные пользователям ДОС. Здесь будет приведен первый из нескольких списков служебных процедур, которые встречаются в этой книге. Данный раздел описывает средства ДОС, а последующие главы будут последовательно описывать служебные процедуры, обеспечиваемые системой BIOS в ПЗУ.

Прежде чем перейти к подробному рассмотрению этих средств, определим способ изложения, который мы будем использовать в дальнейшем. Мы будем последовательно описывать эти средства, указывая их идентифицирующие номера и описывая, что каждое из средств делает. Мы не будем рассматривать во всех подробностях способы вызова служебных процедур, поскольку это не представляет большого интереса. Еще одна причина, по которой не описываются способы вызова служебных процедур ДОС, заключается в том, что читателям этой книги предлагается набор интерфейсных программ, которые избавят Вас от утомительных процедур загрузки регистров и установки

флагов.

В тех случаях, когда нужно будет сделать отступление, чтобы сообщить нечто интересное о конкретной процедуре, ее назначении или связи с другими частями IBM/PC, мы будем делать паузу и излагать этот материал. Даже если Вы не собираетесь писать программы, которые будут использовать эти процедуры, Вам все равно полезно будет прочитать этот раздел. Он поможет Вам больше узнать о том, как IBM/PC работает, какими возможностями обладает и как заставить ее выполнять некоторые удивительные задания.

Если Вы пишите программы для IBM/PC, Вам просто необходимо овладеть всеми средствами доступа к служебным процедурам. Чтобы как можно больше облегчить использование этих процедур, я написал полный набор интерфейсных программ, который записан на дискетах, прилагаемых к этой книге.

Все служебные процедуры ДОС и BIOS в ПЗУ приспособлены для их вызова из программ на языке ассемблера. Для каждой из этих процедур я написал на ассемблере специальную интерфейсную программу, которая обеспечивает доступ к процедуре из программ, написанных на Паскале и других языках высокого уровня. Ассемблерные интерфейсные программы для процедур ДОС, описанных в этом разделе, приведены в листинге 4.101.

Во многих случаях очень полезно иметь более гибкое средство, чем просто интерфейсная программа. Не менее полезной может оказаться и какая-нибудь вспомогательная программа. Для этой цели мною включены в пакет многочисленные программы, написанные на Паскале, которые позволяют наиболее полно использовать вызываемые служебные процедуры. Для средств ДОС, описанных в этом разделе, вспомогательные программы на Паскале приведены в листинге 4.102.

Интерфейсные программы на ассемблере и вспомогательные программы на Паскале позволят Вашим программам получить наиболее полный доступ ко всем возможностям IBM/PC.

В ДОС используется семь процедур обслуживания прерываний и еще одна, восьмая, предназначенная для вызова всех функций ДОС. Сначала рассмотрим отдельные процедуры.

Прерывание 32 (шестнадцатиричное значение 20) используется для завершения работы программы. Это нормальный способ завершения работы программы и запроса к ДОС на выполнение необходимых восстановительных операций.

Эти восстановительные операции имеют два аспекта. Во-первых, имеются три служебных прерывания, которые программе разрешается изменять для собственных целей, а после завершения программы ДОС восстанавливает те значения, которые вектора этих прерываний имеют по умолчанию. (См. разделы

посвященные прерываниям с номерами 34, 35 и 36.) Во-вторых, если программа изменила какие-либо данные, хранящиеся на дискете, может оказаться, что часть данных осталась в системных буферах и ДОС записывает эти данные на дискету. Однако, ДОС не закрывает открытые файлы и программа должна сделать это сама, до обращения к прерыванию 32.

Прерывание 33 (шестнадцатиричное значение 21) используется для вызова функций, которые будут рассмотрены несколько позже. Следующие три прерывания, 34, 35 и 36, используются несколько иначе, чем все остальные. В то время как все прерывания ДОС вызываются нашими программами для запуска процудур ДОС, эти три прерывания используются противоположным образом: они вызываются ДОС для запуска процедур в наших программах. Каждое из трех прерываний вырабатывается ДОС в соответствующий момент времени, определенный для каждого из прерываний. Если установить вектора этих прерываний таким образом, чтобы они указывали на процедуры обслуживания прерываний в наших программах, то наши программы будут получать управление в случае возникновения прерывания. Эти три прерывания предназначены для передачи управления программам пользователей при возникновении одного из трех условий.

Прерывание 34, шестнадцатиричное значение 22, вырабатывается при завершении программы. Это прерывание позволяет организовать специальные завершающие процедуры, которые будут вызываться, когда ДОС завершает выполнение наших программ. Это позволяет обеспечить выполнение восстановительных процедур независимо от того, как завершится Ваша программа.

Прерывание 35, шестнадцатиричное значение 23, вырабатывается при нажатиии комбинации клавиш Ctrl-Break. Это позволяет нашей программе перехватывать сигнал окончания (Break), который служит ДОС указанием аварийно завершить нашу программу. Пример такой возможности Вы можете увидеть в действии при использовании редактора EDLIN, поставляемого вместе с ДОС. Учтите, что ДОС не всегда реагирует на нажатие клавиши Ctrl-Break. Это происходит только в тех случаях, когда опрашивается клавиатура и когда пересылаются данные на экран дисплея. Более подробно об этом будет сказано ниже, при рассмотрении функций ДОС, в частности, функции 8.

Прерывание 36, шестнадцатиричное значение 24, возникает в случае ошибки в работе ДОС. В системе предусмотрена проверка условия отклонения от нормального функционирования ДОС, так называемая "критическая ошибка" и прерывание 36 позволяет нашим программам перехватывать управление, когда такое происходит. В настоящее времмя рассматриваются два типа

критических ошибок, хотя в дальнейшем могут появиться и другие. Первая, наиболее частая ошибка, - это "аппаратная ошибка диска".

Такая ошибка возникает, когда дисковод не может правильно работать, даже после трех попыток ДОС выполнить операцию. Вторая критическая ошибка вызывается нарушением копии таблицы размещения файлов в памяти компьютера (эту таблицу мы рассмотрим в следующей главе). Такая ситуация может возникнуть, если программа случайно будет изменять ячейки в нижних адресах памяти.

Следующие за этими тремя прерывания используются более традиционным образом. Прерывание 37, шестнадцатиричное значение 25, используется для чтения секторов дискеты. Тогда как данные из файлов могут считываться с помощью вызовов функций ДОС, это прерывание позволяет читать сектора в любом месте на дискете, независимо от того, являются они частью файла или нет. Необходимо указать область памяти, в которую должна считываться информация, и еще три параметра: дисковод (в виде порядкового номера, где 0 - это дисковод A, а 1, 2 и 3, соответственно B, C и D), число секторов, которые необходимо прочитать, и номер, указывающий первый сектор, который требуется прочитать.

Для этой и следующей процедур сектора идентифициуются последовательными номерами. Нумерация начинается с нуля, что соответствует первому сектору на дискете, который, при обычном способе адресации секторов, был бы первым сектором дорожки 0 на стороне 0. (Сведения о дорожках, адресах и сторонах дискеты приведены в главе 5.) Сектора пронумерованы последовательно на первой (сторона 0) стороне дискеты. Номера имеют диапазон от 0 до 319 (шестнадцатиричное значение 13F), а затем, для двухсторонних дискет, продолжаются с первого сектора второй стороны, в диапазоне от 320 (шестнадцатиричное значение 140) до 639 (шестнадцатиричное значение 27F).

Этот способ нумерации можно представить в виде формулы. Если пронумеровать стороны дискеты как 0 и 1, дорожки от 0 до 39 и сектора от 1 до 8, то искомый номер сектора для этой служебной процедуры будет вычисляться по формуле:

ИНДЕКС = (СЕКТОР - 1) + (ДОРОЖКА * 8) + (СТОРОНА * 320)

Эта служебная процедура возвращает однобайтный код ошибки, в котором каждый бит указывает отдельное возможное условие ошибки. Номера битов, их числовые эквиваленты и смысл ошибки приведены в следующей таблице:

БИТ ЧИСЛОВОЙ ЗНАЧЕНИЕ

		ЭКВИВАЛЕНТ
1-ый	128	Нет ответа от дисковода
2-ой	64	Ошибка поиска (головка чтения/записи не перемещается к нужной дорожке)
3-ий	32	Ошибка управления (см. описание контроллера фирмы "НЭК" в главе 2)
4-ый	16	Ошибка циклического избыточного проверочного кода (CRC), означает ошибку в данных
5-ый	8	Ошибка ПДП (ошибка при прямом доступе к памяти)
6-ой	4	Сектор не найден (либо неправильный номер сектора, либо ошибка форматирования)
7-ой	2	Дискета защищена от записи (при операции записи)
8-ой	1	Не используется

Эта служебная процедура может оказаться очень полезной для чтения секторов той области дискеты, которая отведена для системных целей, например, для чтения загрузочной записи или справочника файлов (см. в главе 5 дополнительные подробности об этих частях дискеты).

Прерывание 38, шестнадцатиричное значение 26, используется для записи сектора на диск, аналогично тому как прерывание 37 используется для чтения секторов. Остальные подробности работы этих двух процедур одинаковы.

Функции, аналогичные двум последним, позволяет выполнять и система BIOS в ПЗУ. В использовании процедур ДОС, впрочем, имеются преимущества - ДОС обеспечивает автоматическое повторение операции и восстановление в случае ошибкм, а также большую гибкость при использовании различных типов дискет.

Две эти процедуры ДОС позволяют за одну операцию читать и писать несколько секторов диска. Преимущества и недостатки чтения/записи по одному сектору за операцию более подробно рассматриваются в главе 7.

Прерывание 39, шестнадцатиричное значение 27, реализует операцию, получившую название "завершить, но оставить в памяти". Эта процедура используется программами, которые должны остаться в памяти после загрузки и выполнения.

Основное назначение такой процедуры состоит в обеспечении возможности загрузки программ, которые смогут служить для обслуживания прерываний и использования другими программами.

ДОС следит за тем, какой объем памяти в нижних адресах используется векторами прерываний и программами ДОС. Любые программы загружаются в область памяти, располагающуюся выше зарезервированной области. При вызове прерывания 39, адрес верхней границы зарезервированной области памяти изменяется и указывает на ячейку, следующую за последней ячейкой программы, которая должна стать резидентной.

Когда эта процедура используется программой, которая должна впоследствии выплнять функции обслуживания прерываний, происходит следующее. Программа должна быть выполнена один раз, что указывает ДОС на необходимость загрузить эту программу и передать ей управление. После этого программа только загружает свой собственный адрес в таблицу векторов прерываний и вызывает прерывание 39. Позднее, как только произойдет прерывание, управление будет передано резидентной программе.

Помимо семи описанных процедур обслуживания прерываний ДОС, имеется 41 функция ДОС. Каждая из этих функций вызывается с помощью прерывания 33,шестнадцатиричное значение
21. Ниже перечислены все эти функции.

Функция номер 0 в точности соответствует прерыванию 32 - она завершает выполнение программы. Очевидно, что нет необходимости обсуждать преимущества того или иного способа осуществления этой операции.

Функция номер 1 используется для чтения одного символа, введенного с клавиатуры, и отображения его на экране дисплея (в режиме "эхо"). Эта функция ожидает нажатия клавиши. Если нажата одна из клавиш специального назначения, например, функциональная клавиша, то ее нажатие преобразуется в последовательность двух отдельных символов, для ввода которых необходимо выполнить функцию дважды. При этом первый из двух символов имеет код CHR$(0), а второй - один из символьных кодов. Более подробно использование этих клавиш и их кодов описано в главе 10 при рассмотрении клавиатуры. Эта же функция позволяет обнаруживать специальную комбинацию клавиш Ctrl-Break, которая используется для прерывания работы программы.

Функция номер 2 используется для выдачи одного символа на экран дисплея. Для обычных, например, алфавитных, символов эта операция выполняется очень просто. В некоторых, особых случаях выполнение происходит иначе. Все это более подробно рассмотрено в главе 8 и приложении 4.

Функция номер 3 используется для ввода одного байта от адаптера асинхронной связи. Эта функция ожидает ввода символа и не возвращает никакой информации об ошибках. Это один из немногих случаев, когда функция ДОС обеспечивает меньший сервис, чем процедуры системы BIOS в ПЗУ. Соответствующие процедуры системы BIOS в ПЗУ описываются в главе 11.

Функция номер 4 передает один байт данных через адаптер асинхронной линии связи и, таким образом, выполняет операцию, обратную функции 3.

Функция номер 5 используется для передачи одного байта данных устройству печати. В отличие от поцедур системы BIOS в ПЗУ здесь нельзя указать на какое устройство печати должен осуществляться вывод - ДОС работает только с одним устройством. Если к компьютеру подключено несколько устройств печати, то вывод будет осуществляться на первое из них. Соответствующие процедуры системы BIOS в ПЗУ описаны в главе 11.

Функция номер 6 довольно любопытна, поскольку она используется как для ввода с клавиатуры, так и для вывода на экран дисплея. Любой символ, кроме CHR$(255) может быть выдан на экран. Если эту функцию вызвать как бы для вывода на экран символа CHR$(255), то тем самым будет инициирован ввод с клавиатуры. Так же как в случае с функцией номер 1, для кодирования специальных клавиш используются двухбайтовые коды.

Но, в отличие от функции номер 1, эта функция не ожидает нажатия клавиши - если ни одна не нажата, она возвращает признак отсутствия ввода. Эта функция не распознает Ctrl-Break особым образом.

Функция номер 7 аналогична функции номер 1, за исключением того, что символы, вводимые с клавиатуры, не отображаются на экране. Так же как функция номер 6, эта функция не отслеживает нажатия клавиши Ctrl-Break.

Функция номер 8 выполняет те же функции, что и функция номер 1, но без отображения вводимых символов на экране. Подобно функции номер 1 и в отличие от функции номер 7, эта функция следит за нажатием клавиши Ctrl-Break.

Вы, вероятно, уже заметили, что функции 1,6,7 и 8 обеспечивают четыре процедуры ввода с клавиатуры с одной из восьми возможных комбинаций ожидания нажатия клавиши, отображения введенных символов на экране и проверки нажатия клавиши Ctrl-Break. Чтобы не путаться в их назначении, удобно представить все вышесказанное в виде следующей таблицы:

Функция	Ожидание	Эхо	Проверка Ctrl-Break
1	да	да	да
6	нет	нет	нет
7	да	нет	нет
8	да	нет	да

Кроме того, функция 6 дублирует еще и функцию вывода на экран. Ясно, что эти процедуры реализуют базовые операции, которые затем используются командами ДОС, и они вовсе не претендуют на логическую законченность при обслуживании клавиатуры.

Функция номер 9 используется для вывода на экран дисплея строки символов, причем каждый из символов выводится так же, как это делает функция 2. Определенная странность работы этой функции, которая уменьшает ее полезность, заключается в том, что вместо использования счетчика символов в строке, в качестве ограничителя используется знак денежной единицы, "$". Это лишний раз демонстрирует, что ДОС разрабатывалась не как универсальная система, а как средство для решения определенных специальных задач.

Функция номер 10, шестнадцатиричное значение A, обеспечивает буферированный ввод с клавиатуры. В то время как остальные функции ввода немедленно передают введенный символ,
эта функция накапливает законченную логическую единицу ввода, которая завершается нажатием клавиши "ввод" (ENTER). Остальные функции ввода с клавиатуры позволяют программам немедленно реагировать на введенные символы. Однако им присущ такой недостаток как необходимость программного выполнения любых операций редактирования, например, интерпретации клавиши возврата на одну позицию, означающей стирание предыдущего символа. Функция номер 10 передает все стандартные средства редактирования ДОС в распоряжение Вашей программы, что может оказаться очень удобным. Каждую из функций нужно использовать в своем случае, что позволит Вам наиболее эффективно пользоваться возможностями, представленными в Ваше распоряжение операционной системой. Чтобы воспользоваться функцией 10, в Вашей программе должен быть предусмотрен буфер ввода, размер которого может изменяться. Первый байт указывает ДОС размер буфера, а второй байт используется ДОС, чтобы сообщить Вам, сколько символов находится в буфере.

Функция номер 11, шестнадцатиричное значение B, используется для проверки, нажата ли какая-нибудь клавиша на

клавиатуре, без ее ввода. Подобно функции 6, эта функция не ожидает пока будет нажата клавиша. Но, в отличие от функции 6, если клавиша нажата, ее код не вводится и производится проверка нажатия клавиш Ctrl-Break.

Функция номер 12, шестнадцатиричное значение C, очищает буфер ввода с клавиатуры от всех введенных в него символов. Может так случиться, что пользователь начнет набирать символы на клавиатуре прежде чем начнется ввод. Если программа обнаруживает ошибку и посылает пользователю сообщение об ошибке, необходимо иметь уверенность в том, что введенные с клавиатуры символы являются ответом на сообщение об ошибке, а не остатком чего-либо введенного раньше. Для этого и служит функция 12, очищающая буфер ввода. (Пояснение того, как работает буфер ввода можно найти в главе 10). Эта функция обычно предшествует выполнению функций 1,6,7,8 и 10. Буфер, о котором идет речь, является внутренним буфером, а не буфером программы, использующимся в функции 10.

Далее следуют функции обслуживания дискет. Чтобы лучше понять, о чем здесь пойдет речь, необходимо обратиться к информации об организации файлов на дискетах, имеющейся в главе 5. Для использования большинства из этих функций необходим блок управления файлом (FCB) ДОС, который мы здесь описывать не будем.

Функция номер 13, шестнадцатиричное значение D, используется для сброса дисковой подсистемы. Если был изменен номер дисковода, выбираемого по умолчанию, им снова делается дисковод A. См. также функцию 25.

Функция номер 14, шестнадцатиричное значение E, используетяс для установления номера дисковода, выбираемого по умолчанию. Дисковод задается номером от 0 до 3, что соответствует идентификаторам от A до D. См. также функцию 25.

Функция номер 15, шестнадцатиричное значение F, используется, чтобы открыть дисковый файл. Ваша программа должна сформировать блок управления файлом стандартного формата ДОС, включающий имя файла. Если файл не найден на дискете, то программе возвращается признак ошибки. Если необходимо в случае отсутствия файла с указанным именем создать новый файл с таким именем, следует использовать функцию 22. Фактическое открытие файла устанавливает логическое соединение между элементом справочника файлов и блоком управления файлом в программе.

Функция номер 16, шестнадцатиричное значение 10, используется, чтобы закрыть файл, который был открыт функцией 15. Практический смысл функции закрытия файла заключается в модификации элемента справочника файла на дискете.

Функция номер 17, шестнадцатиричное значение 11,

используется для запуска поиска в справочнике файлов дискеты. Эта функция используется при указании неполных или родовых имен файлов, таких как "B:*.BAK" или "???QQ.INC". Если найдено подходящее имя, тогда ДОС сформирует полный блок управления файлом для этого файла, что позволит Вашей программе открыть файл. Использование родовых имен повышает гибкость и расширяет возможности работы с файлами.

Функция номер 18, шестнадцатиричное значение 12, используется для продолжения поиска файлов по родовому имени, начатого функцией 17. Функция 17 запускает поиск и возвращает первое из подходящих имен. Функция 18 продолжает поиск и возвращает имена всех последующих подходящих файлов.

Функция номер 19, шестнадцатиричное значение 13, используется для удаления файла с дискеты. Заданное имя файла может быть родовым, так что не требуется использовать функции 17 и 18 для удаления группы файлов.

Функция номер 20, шестнадцатиричное значение 14, используется для последовательного чтения файла. Из файла считывается очередная запись. Если чтение завершилось неудачей, то передается признак ошибки.

Функция номер 21, шестнадцатиричное значение 15, используется для последовательной записи в файл. Если операция записи прошла неудачно, то передается признак ошибки.

Эти функции последовательного чтения и записи, а также функции прямого чтения и записи, 33 и 34, описанные ниже, используются только для записей фиксированной длины - ДОС не обеспечивает средств для работы с записями переменной длины, например, строками в текстовом файле. Для работы с записями переменной длины Ваши программы должны осуществлять логическую обработку записей. Одно из лучших средств для этого заключается в описании файла в терминах ДОС как последовательного файла с записями длиной в один байт, которые Вы сможите читать и писать посимвольно.

Функция номер 22, шестнадцатиричное значение 16, используется для "создания" файла. Если файл с таким именем существует, то он используется повторно; если нет - то создается новый элемент справочника файлов с нулевой длиной данных. Эта функция также открывает файл, так что ее можно использовать вместо функции 15, если файл необходимо открыть, независимо то того, сущетвует файл или нет. Функция 23, шестнадцатиричное значение 17, используется для переименования одного или нескольких файлов. Для этой функции также могут использоваться родовые имена.

Функции с номером 24 не существует. Функция номер 25, шестнадцатиричное значение 19, используется для определения номера дисковода, использующегося сейчас по умолчанию.

Числовой код дисковода от 0 до 3 соответствует именам от A до D. См. также функции 13 и 14.

Функция номер 26, шестнадцатиричное значение 1A, используется для установления адреса рабочей области диска, DTA, в которой будут выполняться операции с дискетой. Если Ваши программы не заменяют область DTA, устанавливаемую по умолчанию, в ней будет всего 128 байт и нельзя будет пересылать записи большей длины. DTA - это не то же самое, что буфер чтения/записи секторов, используемый ДОС.

Функция номер 27, шестнадцатиричное значение 1B, используется для определения адреса памяти, по которому находится таблица размещения файлов текущего дисковода и другой информации о дискете. Эта функция может быть использована для просмотра таблицы размещения файлов, например, чтобы установить сколько места осталось на дискете. Начиная с версии 2.00 ДОС эта функция обеспечивает доступ только к первой части ТРФ, которая определяет тип дискеты.

Функции с номерами от 28 до 32 не существуют. Функция номер 33, шестнадцатиричное значение 21, используется для чтения произвольной записи, указываемой логическим номером записи. ДОС вычисляет положение произвольной записи по фиксированной длине записи и ее номеру.

Функция номер 34, шестнадцатиричное значение 22, используется для произвольной записи на дискету.

Функция номер 35, шестнадцатиричное значние 23, используется для определения размера файла. При использовании этой функции имеет место ряд странностей. Можно использовать родовое имя файла. Файл не должен быть открыт. Размер файла определяется в терминах длины записи, указанной в блоке управления неоткрытым файлом. Чтобы получить размер файла в байтах, необходимо установить размер записи равным одному байту.

Функция номер 36, шестнадцатиричное значение 24, используется при переходе от последовательного режима к произвольному. Она устанавливает поле произвольной записи в ту же позицию, в которой находится текущая последовательная запись.

Следующие две функции не относятся к обслуживанию дискет.

Функция номер 37, шестнадцатиричное значение 25, используется для установки вектора прерывания. Поскольку вектора прерыванмй устанавливаются очень просто, непонятно для чего нужна эта функция - разве что для обеспечения дополнительной степени защиты.

Функция номер 38, шестнадцатиричное значение 26,

используется для создания нового программного сегмента в порядке подготовки к запуску зависимой программы.

Теперь мы снова вернемся к функциям, обслуживающим файлы на дискетах. Функция номер 39, шестнадцатиричное значение 27, используется для чтения нескольких записей, начиная с произвольной позиции в файле.

Эта функция может быть использована вместо нескольких отдельных вызовов функции 33, если объем памяти позволяет считывать целые блоки из файлов.

Функция номер 40, шестнадцатиричное значение 28, используется при записи в файл, начиная с произвольной позиции, нескольких записей.

Функция номер 41, шестнадцатиричное значение 29, используется для преобразования имени файла в том виде, в каком оно вводится пользователем, например, "B:CHAPTER.4", в формат блока управления файлом, который используется ДОС. Это несомненно очень полезная функция, которая позволяет вводить имена файлов в обычном виде и возлагать всю работу по разбору и преобразованию на ДОС. Если необходимо, эта функция позволит Вам задать идентификатор дисковода, имя файла или расширение имени по умолчанию, когда они не указаны в анализируемой строке. Эта функция также позволяет преобразовывать родовые имена файлов, включающие звездочку в формат родового имени, состоящего из вопросительных знаков, как того требует формат блока управления файлом.

Функция номер 42, шестнадцатиричное значение 2A, используется для считывания системной даты. Дата выдается в виде трех двоичных чисел, соответствующих году, месяцу и дню месяца. Как мы увидим в главе 11, дата автоматически модифицируется при первом же запросе даты или времени после очередной полуночи. Это происходит при любом запросе времени, включая модификацию времени создания файла в элементе справочника файлов. Если время не запрашивалось в промежутке от полуночи до полуночи хотя бы один раз, то дата будет неверной.

Функция номер 43, шестнадцатиричное значение 2B, используется для установки системной даты.

Функция номер 44, шестнадцатиричное значение 2C, используется для считывания системного времени. Время выдается в часах, минутах, секундах и сотых долях секунды. Счетчик таймера, использующийся для подсчета системного времени, изменяется примерно 18 раз в секунду, так что значение сотых долей секунды довольно приблизительно. Поскольку таймер должен перезапускаться примерно каждые 0,0546 секунды, его нельзя использовать для измерения очень коротких промежутков времени. Вычисление сотых долей секунды,

однако, выполняется точно по показаниям счетчика таймера и через определенный промежуток времени значения сотых долей секунды будут равномерно распределены в диапазоне от 0 до 99. Это означает, что Ваши программы могут использовать системное время для генерации псевдо-случайных чисел.

Функция номер 45, шестнадцатиричное значение 2D, используется для установки системного времени.

Функция номер 46, шестнадцатиричное значение 2E, используется для управления верификацией записи на дискету. Если верификация разрешена, то каждая операция записи будет проверяться на возможность возникновения ошибки путем повторного считывания данных и их сравнения с исходными данными. Маловероятно, что Вам потребуется эта функция, так как дискеты достаточно надежны и довольно значительные проверки выполняются автоматически. Но, если Вам потребуется верификация, то используйте функцию 46.

Мы перечислили все имеющиеся функции ДОС. Позднее, когда мы рассмотрим все возможности, предоставляемые системой BIOS в ПЗУ, Вы заметите, что некоторые функции дублируются. Однако, в большинстве случаев ДОС и BIOS в ПЗУ обслуживают совершенно различные нужды.

В следующей главе мы начинаем рассмотрение важного предмета - дискет и форматов хранения данных на дискетах.

ГЛАВА 5. ОРГАНИЗАЦИЯ ВНЕШНЕЙ ПАМЯТИ

[Оглавление]

Дискеты стали не только стандартным, но и самым распространенным носителем программ и данных для персональных
компьютеров. В этой главе будут приведены все необходимые сведения о дискетах, об особенностях их структуризации. Кроме того, вы сможете кое-что узнать о защите от копирования.

Весь материал, приведенный в настоящей главе, относится к семейству компьютеров IBM/PC, совместимыми с ними компьютерами, а также ко всем компьютерам, работающим под управлением операционной системы MS-DOS. Другие компьютеры могут использовать другие размеры дискет и форматы данных.

Как уже указывалось в предыдущей главе, начиная с версии ДОС 2.00, подключение к системе дискет новых типов осуществляется довольно просто. Схема выполнения этой процедуры предполагает, что новые типы дисков используются таким же образом, как те дискеты, которые были исходно доступны для IBM/PC. Следовательно, вся информация, изложенная в этой главе, может применяться и к тем новым

дисковым устройствам, которые будут добавлены в Вашу систему, за исключением конкретных чисел, задающих специфические функции.

5.1. Устройство гибкого машнитного диска (дискеты)

Гибкие диски, как известно, являются идеальным носителем для внешней памяти персональных компьютеров, подобных IBM/PC. Их размеры удобны, цена не высока и они достаточно надежны в эксплуатации. При известной осторожности в обращении дискеты очень редко портятся. Именно по этому дискеты стали наиболее популярным носителем информации задолго до появления компьютеров IBM/PC. Вполне естественно, что фирма ИБМ обратилась к дискетам, выбирая носитель информации для своего компьютера IBM/PC.

Кратко рассмотрим, что собой представляет сама дискета. (Простая схема приведена на рис.5.1.) Дискета представляет собой круглый кусок гибкого пластика, покрытый магнитным окислом, напоминающим покрытие других известных магнитных носителей, например, магнитных лент. Магнитные диски, использующиеся на больших компьютерах, изготавливаются из жестких металлических пластин, а для дискет используются гибкие пластиковые кружки, что и дало им популярное название "гибкие" или "флоппи" - диски. То, что эти диски были сделаны гибкими, значительно уменьшило вероятность их повреждения при обращении с ними и это в значительной мере определило их успех.

Круглый диск с магнитным покрытием всегда помещен в квадратный предохранительный конверт. Внутренняя поверхность этого конверта покрыта слоем белого фетро-подобного материала, помогающим в защите дискеты. Он служит как для смягчения ударов, так и для улавливания пыли.

Квадратный предохранительный конверт, который по неизвестной причине практически всегда имеет черный цвет, имеет четыре отверстия, каждое из которых имеет свое собственное назначение. Отверстие в центре предназначено для захвата дискеты приводом дисковода. Через это отверстие механизм дисковода захватывает гибкий диск, чтобы привести его во вращение.

Второе отверстие в предохранительном конверте представляет собой продолговатую прорезь, через которую осуществляется доступ к дискете головок чтения/записи. Через эту прорезь осуществляется процесс чтения и записи информации.

Третье отверстие представляет собой небольшую дырочку

возле центрального отверстия. Это отверстие в конверте позволяет следить за небольшим индексным отверстием в самой дискете. Индексное отверстие используется для указания начала и конца дорожки на дискете. Это индексное отверстие служит для определения точки отсчета при чтении или записи информации.Если Вы никогда не видели индексного отверстия, осторожно поверните дискету внутри конверта, пока оно не совместится с отверстием в конверте. В дискетах такого типа, который используется в IBM/PC, это единственное отверстие в пластиковом кружке.

Четвертое отверстие - квадратная прорезь на краю предохранительного конверта - служит в качестве признака защиты записи. Практически все магнитные носители используют те или иные признаки защиты от записи. Если эта прорезь открыта, то запись на эту дискету может выполняться дисководом, если же она закрыта, то дискета защищена от записи. В конвертах некоторых дискет вообще нет такой прорези, что полностью исключает возможность случайного стирания записанных на ней данных. Отсутствие прорезей защиты записи характерно для многих дискет с программным обеспечением IBM/PC, в частности, с самой операционной системой ДОС. Эти дискеты записываются на специальных дисководах, игнорирующих признак защиты записи.

Не следует, однако, считать, что заклеив прорезь защиты записи Вы полностью защитите свои дискеты от случайного стирания. Закрытая прорезь говорит только о том, что правильно работающий дисковод не будет пытаться выполнять запись на такую дискету. Неисправный дисковод может делать все что угодно, независимо от наличия или отсутствия признака защиты. Конечно, маловероятно утратить данные именно таким образом, но все же это может произойти.

5.2. Форматы хранения информации

Данные на дискете размещаютя по "дорожкам". Дорожка - это замкнутая окружность на поверхности дискеты, размещенная на определенном растоянии от центрального отверстия. Стандартный формат дискеты для IBM/PC имеет 40 дорожек.

Каждая дорожка разделена на части, называемые "секторами" или "записями". Сектора представляют собой основную единицу хранения информации на дискете. При чтении или записи устройство считывает или записывает всегда целый сектор, независимо от объема запрашиваемой информации несмотря на отсутствие огрничений. Хотя Вы, точнее Ваши программы, могут организовывать свои данные в единицах произвольного размера, компьютер и дисковод всегда работают с

секторами, что, как правило, скрыто от пользователя.

Термин "запись" иногда используется в том же смысле, что и "сектор". Лучше будет оставить слово "запись" для обозначения логических единиц хранения информации в наших программах и использовать слово "сектор" для обозначения физической единицы хранения информации на дискете.

Данные могут размещаться как на одной стороне дискеты, так и на двух сторонах. Схема размещения секторов на дискете показана на рис. 5.2.

Сейчас нужно ненадолго прерваться и договориться о терминологии. Дискета имеет одну или две стороны; на каждой стороне по 40 дорожек; по восемь секторов на дорожке и по 512 байт в каждом секторе. Как к ним обращаться? В этой книге будет использоваться способ нумерации, принятый во всех руководствах фирмы "ИБМ", хотя он и несколько противоречив. Стороны нумеруются как 0 и 1, односторонняя дискета имеет только сторону с номером 0 . Дорожки нумеруются с 0 по 39 ; причем дорожка 0 находится ближе всего к внешнему краю дискеты, а дорожка 39 ближе к центру. Начало дискеты находится на нулевой дорожке. Сектора, похоже просто чтобы сделать их в чем-то отличающимися, нумеруются с 1 по 8. Таким образом, первый сектор диска имеет номер 1 и располагается на нулевой дорожке нулевой стороны.

Данные внутри сектора размещаются и адресуются произвольно. Если пользоваться смещениями, то для первого байта смещение будет равно нулю, для второго - единице, и так далее. В дальнейшем мы будем обращаться к данным как по смещению (начиная с 0), так и по порядковому номеру (начиная с 1), как в конкретном случае будет удобнее.

Сектора на дискете имеют определенную последовательность и, когда это возможно, они используются в этой последовательности. Последовательность начинается с самого первого сектора на дискете, за которым следуют остальные сектора этой же дорожки (нулевой) со второго по восьмой. У двухсторонних дискет за восьмым сектором нулевой стороны следует первый сектор той же дорожки, но первой стороны. Как при одностороннем, так и при двухстороннем формате дискет за последним сектором одной дорожки следует первый сектор следующей дорожки.

Вы, вероятно, уже заметили, что на двухсторонних дискетах прежде чем переходить к очередной дорожке, изменяется сторона, так что используется одна дорожка на обеих сторонах, прежде чем система перейдет к следующей дорожке. Таким образом, получается, что лучше было бы использовать при обращении к конкретному сектору

последовательность дорожка-сторона-сектор, а не - сторона-дорожка-сектор, поскольку они используются именно в таком порядке. Тем не менее, во всех руководствах фирмы "ИБМ" обычно указывают сектор, задавая сторону, потом дорожку, а потом номер сектора и мы тоже будем пользоваться такой же последовательностью.

5.3. Типы дмскет и проблемы защиты от копирования

Типов дискет насчитывается значительно больше, чем используется семейством компьютеров IBM/PC. Рассмотрим эти типы, чтобы лучше понимать, что именно используется для IBM/PC.

Обычные гибкие диски имеют один из двух размеров. Больший формат имеет диаметр 203 мм (8 дюймов), а меньший - 133 мм (5,25 дюйма). Недавно появившиеся так называемые "микро-флоппи", диаметром 89 мм (3,5 дюйма), заключены в твердую упаковку и, строго говоря, не являются гибкими дисками.

Компьютеры IBM/PC, подобно большинству персональных компьютеров, используют мини-диски диаметром 133 мм (5,25 дюйма), а компьютер IBM Displaywriter (довольно близкий к PC по своим характеристикам) использует больший формат.

Еще одной характеристикой дискеты, является плотность записи. В сфере персональных компьютеров используются различные плотности записи, а существующая терминология не всегда понятна. Стандартные дискеты, используемые IBM/PC, имеют плотность записи, называемую двойной, что позволяет записывать и считывать 40 дорожек данных на поверхности дискеты. Другой использующийся формат называется учетверенной плотностью, что на том же пространстве позволяет иметь 80 дорожек. Иногда плотность записи обозначается в количестве дорожек на дюйм. Двойная плотность, нормальная для IBM/PC, составляет 48 дорожек на дюйм (48 TPI), а учетверенная плотность составляет 96 дорожек на дюйм (96 TPI).

Существует два способа разбивки (разметки) дорожек на сектора. Один из них называется фиксированным или аппаратным, а второй - программным. Если размер сектора задан жестко и определяется механическими характеристиками устройства, то такая разметка называется фиксированной. При фиксированной разметке индексные отверстия, расположенные по кругу, обозначают начало каждого сектора и, следовательно, его положение на дискете точно определено. Альтернативой для такой разметки служит программная разметка.

Для стандартных дисков персональных компьютеров, размером 133 мм (5,25 дюйма), расположение дорожек на диске и

число сторон определяются характеристиками самих дисков и дисководов и, по существу, являются неизменными. Однако, количество секторов на дорожке и их размер определяются программно, в процессе форматирования. Именно поэтому гибкие диски называют дисками с программной разметкой секторов (soft-sector). Форматирование (определение характеристик секторов) выполняется либо программами операционной системы, либо базовой системы ввода/вывода (BIOS).

Размер сектора 133 мм диска, поддерживаемого системой BIOS, может составлять 128, 256, 512, и 1024 байта. ДОС (версий 1.XX-3.1) ориентирована на использование секторов, размером 512 байт.

В ранних версиях ДОС использовалось ограниченное число дисковых форматов. Начиная с версии 2.00 и во всех последующих используется четыре основных формата. Различие между форматами определяется числом используемых сторон диска
и числом секторов на дорожке: одно- или двухсторонний диск и 8 или 9 секторов на дорожке.

Обозначение	Кол-во сторон	Количество секторов	Количество дорожек	Объем (Кбайт)
S-8	1	8	40	160
D-9	2	8	40	320
S-9	1	9	40	180
D-9	2	9	40	360

Причина существования чеиырех различных форматов проста, фирма хочет быть уверена, что все версии ДОС смогут поддерживать программы, разработанные для любых, самых ранних версий.

Первые модели IBM/PC оснащались односторонними гибкими дисками. Впоследствии начали выпускаться дисководы для двухсторонних дисков.

В ранних версиях ДОС на каждой дорожке размещалось восемь 512-байтных секторов, хотя на той же дорожке можно легко разместить и десять. Позже установили, что достаточно надежно размещать на дорожке девять секторов, после чего девятисекторный формат стал стандартным.

Увеличение числа форматов тесно связано с развитием операционной системы ДОС. Первоначальная версия ДОС поддерживала всего один формат, который я обозначил как S-8. В следующей версии появился формат D-8. В версию 2.00 были

включены девятисекторные форматы S-9 и D-9. В версии 2.10 новых форматов не появилось, а в версию 3.0 был включен формат с учетверенной плотностью записи, который будет кратко рассмотрен несколько позже.

Широкое применение получили только два формата: S-8 и D-9. Формат S-8 является как бы наименьшим общим делителем, он традиционно используется для коммерческих программ, что гарантирует ее считывание в любой версии ДОС.

Форматы с учетверенной плотностью записи

Как читатель уже наверное обратил внимание, все четыре стандартных формата имеют одинаковое количество дорожек. Во всех этих форматах используется по 40 дорожек. Это связано с тем, что дисководы, наиболее часто использовавшиеся в семействе машин IBM/PC, разрабатывались для чтения/записи сорока дорожек. Некоторые дисководы с 133-мм дисками и практически все дисководы для 89-мм (3,5 дюйма) дисков позволяют записывать 80 дорожек. Они получили название устройств с учетверенной плотностью записи. Среди форматов, использующихся такими устройствами, наибольшее распространение получили форматы QD-9 и QD-15.

Обозначение	Количество сторон	Количество секторов	Количество дорожек	Объем (КБайт)
QD-9	2	9	80	720
QD-15	2	15	80	1200

Формат QD-9 отличается от D-9 только удвоенным числом дорожек. Формат QD-9 чаще всего используется не для стандартных 133-мм дисков, а для 89-мм (3,5 дюйма) микро-дисков. Дисководы с учетверенной плотностью записи могут подключаться к обычному компьютеру IBM/PC как нестандартные устройства, если включить в ДОС соответствующий драйвер. Предполагается, что этот формат будет использоваться очень широко в самом ближайшем будущем.

Формат большой емкости QD-15, используемый в компьютере IBM/PC модели AT, имеет в каждой из 80 дорожек по 15 секторов, размером 512 байт. Это стало возможным благодаря использованию в компьютере AT специальных дискет, магнитное покрытие которых отличается от обычного. Только при условии использования этих дисков и специальных дисководов может применяться такой формат.

Формат жесткого диска

Накопители на жестких магнитных дисках большой емкости, такие как 10-мегабайтный жесткий диск компьютера XT или 20-мегабайтный диск компьютера AT, имеют собственные форматы, на которых необходимо остановиться особо.

Любой диск имеет физический и логический формат. Физический формат диска определяет размер сектора (в байтах), число секторов на дорожке (или в цилиндре - для жестких дисков), число дорожек (цилиндров) и число сторон. Логический формат диска определяет способ организации информации на диске и фиксирует размещение информации различных типов.

В отличие от гибких дисков, физический и логический формат которых устанавливается в процессе форматирования дискеты, жесткие диски поступают к потребителю с определенным физическим форматом, который устанавливается в процессе изготовления диска. Логическая структура жесткого диска устанавливается пользователем, причем это должно быть сделано прежде чем начнется использование диска операционной системой.Установка логической структуры диска выполняется в два этапа. Во-первых, жесткий диск можно разбить на части, каждая из которых может использоваться своей операционной системой. Далее, каждую из этих частей необходимо отформатировать в соответствии с требованиями той определенной системы, для которой она предназначена.

Физические форматы жестких дисков:

Тип компьютера	Число сторон	Число секторов	Число цилиндров	Объем (МБ)
XT	4	17	306	10
AT	4	17	615	20

Вернемся теперь к процессу форматирования. Данные на дискетах не могут записываться и считываться произвольно. Необходимо обрамление данных специальными электронными сигналами, записанными на дискете.

Это обрамление называется форматирующим. Поэтому, новые дискеты до их использования должны быть отформатированы командой FORMAT операционной системы ДОС.

Основная часть процесса форматирования состоит в записи адресных меток, идентифицирующих каждый сектор, и указывающих его размер. Изучив более подробно способы доступа к данным, хранящимся на дискете, Вы увидите, что одна из ошибок, возникающих при чтении или записи, состоит в отсутствии адресных меток.

Команда FORMAT не только размечает сектора дискет, формируя адресные метки, но также устанавливает определенное значение в поля даных всех секторов. Это значение (F6/16) или символьный код CHR$(246) позволяет определять, использовался ли когда-нибудь определенный сектор или нет.

Одна из основных причин использования программной разметки заключается, вероятно, в обеспечении средств защиты от копирования. Операционная система ДОС позволяет считывать и записывать только сектора стандартного размера 512 байт, хотя сами дисководы могут работать с секторами произвольного размера. Можно написать такую программу, которая не будет использовать средства ДОС для работы с гибким диском и будет хранить часть своей информации или даже всю информацию в секторах нестандартного размера, которые не будут читаться стандартными программами ДОС. В этом и заключается основной принцип защиты от копирования в IBM/PC. Хотя существуют различные способы защиты от копирования, наиболее простой заключается в использовании секторов нестандартного размера. Обычно, лишь небольшая часть информации хранится на защищенной от копирования дискете в нестандартных секторах, а основная часть данных может храниться в обычных 512-байтных секторах.

Понятно, что все эти различные типы дискет практически несовместимы. Одно из немногих исключений связано с одновременным использованием одно- и двухсторонних дискет. Однако, для совместимости дискет важны даже не физические различия. Для совместного использования дискеты должны иметь совместимую логическую разметку (например, размещение справочника файлов). Из-за логических различий невозможно

просто перенести данные с дискеты компьютера "Эппл" в память компьютера IBM/PC, или из операционной системы ДОС в операционную систему CP/M-86. Для этого нужно написать специальную программу преобразования, которая сможет выполнять эту задачу.

5.4. Стандартный накопитель информации на гибких магнитных дисках

Иногда какой-либо компании удается сделать столь хорошую продукцию, что она становится промышленным стандартом, захватывая существенную часть рынка за счет собственных объемов продаж и продаж близких имитаций этой продукции.Это произошло с серией накопителей на гибких магнитных дисках ТМ 100 фирмы "Тэндон". Судя по всему, фирма "ИБМ" приняла эти дисководы в качестве эталонных и, насколько можно о чем-либо судить в этой сфере тщательно хранящихся секретов, только дисководы серии ТМ 100 фирмы "Тэндон" поставляются фирмой"ИБМ" в качестве официального дисковода для IBM/PC.

Существует четыре популярных варианта дисководов ТМ 100. Рассмотрим эти варианты подробнее.

Устройства ТМ 100-1 предназначены для работы с односторонними дисками с двойной плотностью записи (т.е., с 40 дорожками). Это устройство входило в стандартный комплект первой версии IBM/PC. Позднее, одновременно с внедрением ДОС версии 1.10, фирма "ИБМ" начала использовать дисководы ТМ 100-2, работающие по-прежнему с двойной плотностью, но уже с двухсторонними дисками, что позволило вдвое увеличить объем хранения информации. Ниже мы рассмотрим, как операционная система ДОС использует разные типы дискет). Оставшиеся две модели, ТМ 100-3 и ТМ 100-4, устройства позволяющие работать с одно- и двухсторонними дисками с учетверенной плотностью записи, т.е. с восьмьюдесятью дорожками на поверхности дискеты.

Не совсем понятно решение фирмы "ИБМ" использовать односторонние дискеты наряду с двухсторонними. С точки зрения дальних перспектив компьютера такое решение выглядит неразумным. Наиболее вероятной причиной использования односторонних дискет является желание фирмы ИБМ снизить цену своего компьютера, которая и так существенно превышает цену обычных домашних компьютеров.

Имеется ряд признаков, свчидетельствующих о стремлении фирмы предельно снизить цену базовой модели. По крайней мере, это единственный аргумент которым можно объяснить появление практически нигде не использующейся модели с накопителем на

кассетной магнитной ленте и применения односторонних дискет в первых модификациях IBM/PC. Расчет делался на то, что серьезные пользователи, думающие о перспективе, будут приобретать машины, оснащенные более мощными дисками, а чтобы
не отпугнуть колеблющегося новичка высокой ценой, фирма ИБМ выбрала из всех имеющихся дисководов устройство с наиболее умеренной ценой - устройство с односторонними дисками.

5.5. Принципы хранения файлов

Можно использовать различные схемы для организации, хранения и учета данных на дискете. Каждая из схем имеет свои достоинства и недостатки с точки зрения эффективности использования пространства памяти дискеты, скорости доступа, безопасности и качества хранения данных. (Понятие качества хранения данных подразумевает вероятность каких-либо нарушений при хранении и трудность восстановления данных, если нарушения произошли.) Разработка схемы хранения данных состоит в искусстве сочетания всех этих аспектов, иногда противоречащих друг другу.

В этом параграфе будет описана схема хранения данных, используемая в MS-ДОС. Прежде чем перейти к подробному рассмотрению схемы хранения, опишем ее в общих чертах. Первое, что следует подчеркнуть, это использование для хранения данных стандартных 512-байтных секторов. Второе. Все сектора извлекаются из общего пула свободных секторов - никакие сектора или области на дискете не резервируются (исключение составляет область для размещения системных файлов IBMBIO.COM и IBMDOS.COM, о чем будет сказано ниже). Третье. Распределение секторов данных и обьединение секторов, образующих файл данных, выполняется независимо от обслуживания самих секторов данных, с помощью механизма, получившего название Таблицы Распределения Файлов (ТРФ). И последнее. Каждая дискета имеет справочник или таблицу гсодержимого, котороая служит для учета файлов, хранящихся на дискете файлов. Описанная схема хранения данных предполагает существование четырех различных типов секторов, один из которых используется для хранения данных, а три других имеют специальное назначение.

Как уже упоминалось в предыдущей главе, любая дискета, которая должна использоваться для запуска операционной системы, должна иметь программу загрузки, записывающуюся в самом первом секторе дискеты. Программа загрузки всегда записывается в первый сектор любой форматируемой дискеты

(если задан сообтветствующий режим форматирования) независимо от того, будет ли она когда-нибудь использоваться для запуска системы. Сектор с программой загрузки представляет собой первый специальный тип сектора в ДОС.

Второй специальный тип сектора используется для хранения таблицы таблиы размещения файлов (ТРФ). ТРФ занимает два сектора, следующих за сектором с программой загрузки. Таблица размещения файлов служит для индикации занятости секторов данных на дискете.

Третий и последний специальный тип сектора используется для хранения справочника дискеты. Справочник располагается следом за ТРФ. Справочник может иметь различные размеры: односторонние дискеты имеют четыре сектора справочника, а двусторонние дискеты - семь.

Все эти сектора специального назначения занимают семь первых секторов односторонней дискеты или десять - двусторонней. Все остальные сектора используются для хранения данных. На рис. 5.3 показано размещение всех этих четырех типов секторов на дискете.

Справочник и таблица размещения файлов располагаются в начале дискеты. На первый взгляд это представляется оптимальным размещением. Однако, при доступе к файлу ДОС сначала должен найти элемент этого файла в справочнике, а потом обратиться собственно к данным на дискете. В среднем, расстояние между справочником и собственно файлом на дискете составляет 20 дорожек, то есть, практически половину дискеты. Перемещение головок чтения-записи в дисководе является самой медленной операцией. Так что расстояние между справочником и самим файлом может иметь важное значение с точки зрения временных характеристик работы.

Если бы справочник располагался в середине дискеты, то среднее расстояние до секторов данных уменьшилось бы вдвое, т.е., до 10 дорожек. С другой стороны, работа с пространством данных, состоящим из двух частей, по обе стороны от справочника, будет значительно сложнее. Для операционных систем персональных компьютеров выгода от размещения справочника посредине дискеты слишком мала, чтобы браться за разрешение возникающих при этом дополнительных проблем. Такой прием обычно используется в больших компьютерных системах.

Далее мы подробно рассмотрим справочник дискеты и таблицу размещения файлов. Разобравшись в этом вопросе нам легче будет понять, как производится распределение пространства на диске между файлами.

5.6. Организация справочника

Справочник дискеты содержит список всех файлов, находящихся на дискете. Элементы справочника содержат всю необходимую информацию о файле, за исключением информации о
размещении файла (которая хранится в ТРФ).

Каждый элемент справочника имеет длину 32 байта. Следовательно, в 512-байтном секторе помещается ровно 16 элементов справочника. На односторонней дискете выделено 4 сектора для справочника, что позволяет хранить 64 элемента справочника. Семь секторов справочника двусторонней дискеты позволяет разместить 112 элементов.

Большая часть из 32 байтов элемента справочника используется для хранения информации о файле, однако имеется ряд неиспользуемых полей, зарезервированных для дальнейшего использования. Одно из различий между версиями ДОС 1.00 и 1.10 заключается в использовании некоторых из этих полей, а ДОС версии 2.00 еще более расширяет использование справочника.

Теперь подробно рассмотрим каждую часть элемента справочника.Листинг программы 5.1 содержит об~явления форматов данных на языке Паскаль, а листинг 5.3 содержит несколько процедур для работы со справочником - чтения, интерпретации и записи обратно на дискету.

Программа DiskLook, входящая в пакет программ, прилагаемых к этой книге, разработана для отображения полной информации из справочника дискеты. Во время изучения этой главы можно использовать эту программу, чтобы лучше понять назначение всех полей элемента справочника. Если используется программа DiskLook, то для получения на экране информации из справочника нужно нажать функциональную клавишу F4.

Каждый элемент справочника состоит из восьми частей или полей следующего назначения:

ИМЯ ФАЙЛА: Это поле имеет длину восемь байт, расположенных в элементе справочника со смещениями от 0 до 7 и содержит имя файла. Если длина имени меньше восьми символов, то недостающие символы заменяются пробелами.

Правилами оформления имен файлов в ДОС запрещается включать пробелы в середину имени файла. Но такие имена файлов все же могут создаваться. Это можно делать на языке Бейсик, поскольку в нем ограничителем имени служат кавычки. Например, следующий оператор Бейсика позволяет создать файл, имя которого содержит пробел:

```
OPEN "AA BB.FIL"
```

Программа "IBM Typing Tutor", написанная на Бейсике, использует файлы, имена которых содержит пробелы. При использовании программы поиска имени файла в справочнике, не следует считать, что имя файла завершается первым же встреченным пробелом - имя может продолжаться и после этого пробела.

Если первый байт имени файла имеет шестнадцатеричное значение E5, CHR$(229), это означает, что элемент справочника не используется. Это может значить, что он никогда не использовался или что файл, которому соответствовал этот элемент, уже уничтожен. Проверка второго байта имени позволяет установить отличие неиспользовавшегося элемента от элемента уничтоженного файла (если значение второго байта больше символьного кода "Z", этот элемент не использовался). В последних версиях ДОС неиспользованные элементы справочника
содержат в первом байте 0.

После уничтожения файла с ним происходит следующее: занимаемое им место возвращается в общий пул (об этом будет подробнее сказано в следующем разделе) и первый символ имени файла заменяется шестнадцатеричным кодом E5, CHR$(229). Вся информация в элементе справочника, за исключением первого символа имени, сохраняется. Благодаря этому, программа DiskLook может отображать имена уничтоженных файлов, а программа подобная UnErase - восстанавливать файлы.

Еще один код, который может встретиться в первом байте имени файла - это шестнадцатеричный код 2E, который указывает на справочник нижнего уровня.

РАСШИРЕНИЕ: Это поле имеет длину три байта, которые расположены со смещениями от 8 до 10 от начала элемента справочника. Оно содержит расширение имени файла. Как и в случае с самим именем, короткое расширение дополняется пробелами. Если файл не имеет расширения, то это поле содержит три пробела.

АТРИБУТ: Это поле состоит из одного байта, расположенного со смещением 11 от начала элемента справочника. Поле атрибута используется для установления признака "скрытого" файла, т.е. такого файла, имя которого не обнаруживается обычными программами работы со справочниками. Два бита этого байта служат для индикации атрибутов "скрытого" и "системного" файла. Остальные шесть бит не были определены в ранних версиях ДОС и только начиная с версии 2.00 часть из них стала использоваться. Во всех более ранних версиях эти биты не определены и устанавливаются в нуль.

Седьмой бит байта атрибутов с числовым значением 2

определяет скрытый файл, а шестой, с числовым значением 4, - системный файл. Таким образом, обычный видимый файл будет иметь нулевой байт атрибута, для скрытого файла байт атрибута имеет значение 2, для системного файла - 4, а для скрытого системного файла - 6.

Хотя обработка системного атрибута осуществляется независимо от скрытого, оба эти атрибута практически совпадают по своему функциональному назначению. При использовании любого из них файл становится "невидимым" для обычных программ работы со справочниками. Системный атрибут зарезервирован не для использования в будущем, а (по сведениям фирмы "Майкрософт") для обеспечения совместимости с другими операционными системами.

В версиях ДОС, следующих за версией 2.00 используются еще четыре бита. Бит 0 (с единичным значением) указывает, что файл доступен только для чтения. Такой файл защищен от изменений и уничтожения средствами ДОС.

Бит 3 (с числовым значением 8) указывает, что элемент справочника содержит метку тома. Сама метка хранится в полях имени файла и расширения, которые воспринимаются в этом случае как одно целое.

Бит 4 (с числовым значением 16) используется для указания элементов справочника, соответствующих справочникам нижнего уровня. Поскольку справочники нижнего уровня хранятся на диске подобно обычным файлам данных, им необходим собственный элемент в корневом справочнике. В этом элементе используются все поля, кроме размера файла, в данном случае равного нулю. Действительный размер файла справочника нижнего уровня легко определяется из соответствующей последовательности в ТРФ.

Бит 5 (с числовым значением 32) предусмотрен для облегчения создания резервных копий файлов на жестких дисках. Для файлов на гибких дисках этот атрибут практически бесполезен.

ЗАРЕЗЕРВИРОВАННОЕ ПОЛЕ: Это поле зарезервировано для возможного использования в будущем. В первой версии ДОС это поле имело длину 12 байт, но в дальнейшем его сократили до 10 байт, расположенных со смещениями от 12 до 21 относительно начала элемента справочника.

Любые новые операции над справочником файлов, которые могут использовать это поле. Когда элемент справочника находится в активном состоянии (используется), то эти байты принимают значение 00. В исходном состоянии байты этого поля имеют шестнадцатеричные значения F6, установленные командой FORMAT. Любые другие значения этого поля указывают на какой-либо вариант его использования.

ВРЕМЯ: Это поле имеет длину два байта в формате 16-разрядного целого числа без знака, расположенные со смещениями 22 и 23 относительно начала элемента. Начиная с версии ДОС 1.10 в этом поле хранится время создания или модификации файла. В первой версии ДОС хранилась только дата, а это поле было частью зарезервированного поля.

Большинство операций, использующих это поле, таких как операция распечатки содержимого справочника DIR, выдают время с точностью до минут, хотя число хранящееся в поле времени позволяет определить время с точностью до двух секунд.

Код времени, хранящийся как 16-разрядное целое число без знака, вычисляется по следующей формуле:

время = часы * 2048 + минуты * 32 + секунды / 2

ДАТА : Это поле состоит из двух байт, хранящихся со смещениями 24 и 25 от начала элемента. Как и время, даты хранятся в виде целого 16-разрядного числа без знака, которое вычисляется по формуле:

Дата = (год - 1980) * 512 + Месяц * 32 + День

Диапазон изменения лет составляет от 1980 до 2099, причем хранятся они в виде относительных величин от 0 до 199. Хотя формат позволяет задавать относительный номер года 127 (что соответствует 2107 году), программы ДОС позволяют работать с годами только до 2099. Никто, правда, не ожидает, что ДОС будет использоваться так долго.

Как формат, так и размещение полей даты и времени подобраны таким образом, чтобы вместе они образовывали единое четырехбайтовое поле, которое можно использовать в операциях сравнения. Достаточно просто извлекать компоненты даты и времени из соответствующих полей и вычислять их разность. Например, для разделения даты на составные части можно использовать следующие формулы, записанные на Паскале:

```
год := 1980 + поле_даты div 512
месяц := (поле_даты mod 512) div 32
день := поле_даты mod 32
```

НОМЕР НАЧАЛЬНОГО КЛАСТЕРА : Это двухбайтовое поле, расположенное со смещением 26, 27 от начала элемента, содержит 16-разрядное число, являющееся смещением до начальной точки файла в таблице размещения файлов (ТРФ); подробнее от этом сказано в следуцющем разделе. Начальный кластер является первой частью пространства данных на дискете.

РАЗМЕР ФАЙЛА : Это поле состоит из четырех байтов, размещенных со смещениями от 28 до 31 от начала элемента. Размер файла задается в байтах и хранится в формате четырехбайтового целого числа без знака, которое может рассматриваться как пара двухбайтовых чисел. Размер файла не всегда указывает точное число байтов. Для всех файлов это поле должно соответствовать размеру файла в секторах.(Если по какой-либо причине это не так, то процедура ДОС CHKDSK сообщит об ошибке и установит правильную длину.)

Для программных файлов, представленных в формате типа ".COM" и для файлов, созданных из данных фиксированной длины, только поле размера файла позволяет точно определить, где находится конец данных. Для этих файлов значение в поле размера файла хранится с точностью до байта.

Для файлов некоторых других форматов такая точность не обязательна и размер файла, указанный в соответствующем поле, может отличаться от действительного. Наиболее часто это случается с текстовыми файлами, которые будут описаны в разделе 5.9 . Текстовые файлы в коде ASCII имеют маркер (признак) конца файла, хранящийся в самом файле, который фиксирует точный конец данных. По этой причине многие программы, работающие с текстовыми файлами, не слишком заботятся о сохранении действительного размера файла.

Например, если программа формирует текстовой файл не побайтно, а блоками по 128 байт, то ДОС определит размер файла с точностью до 127 байт. Для программ редактирования текстов значительно эффективнее читать и записывать данные большими блоками, чем делать это побайтно. Такой подход является довольно распространенным и не следует чересчур доверять значению размера файла, по крайней мере, для текстовых файлов.

Справочники нижнего уровня

Существуют два типа справочников: корневые и справочники нижнего уровня. Содержимое и характер использования справочников обоих типов по существу одинаковы - те и другие хранят имена файлов и сведения о них и их расположении на диске. Однако, характеристики этих справочников различаются. Корневые справочники (или просто справочники) имеют фиксированный размер и хранятся в определенном месте на дискете. Справочник нижнего уровня является дополнением корневого справочника и может храниться как обычный файл в любом месте на дискете. Справочники нижнего уровня могут использоваться любой версией ДОС, следующей за версией 2.00.

Справочники нижних уровней хранятся в пространстве данных диска как обычные файлы. Форматы полей и содержимое справочников нижнего уровня такие же как у корневого справочника, с той лишь разницей, что размер справочника нижнего уровня не ограничен. Справочники нижнего уровня всегда связаны с порождающим справочником, который может быть как корневым справочником, так и справочником более высокого уровня.

В порождающем справочнике всегда имеется элемент для справочника нижнего уровня,отличающийся от обычного элемента установленным битом 4 байта атрибутов и нулевым значением размера файла. Действительный размер справочника можно определить, проследив всю цепочку в таблице размещения файлов.

При создании справочников нижнего уровня в них выделяются два элемента, в которых в качестве имени файла указана точка и две точки ("." и "..").

Элемент с одной точкой относится к этому справочнику, а элемент с двумя точками - к его порождающему справочнику. Номера начальных кластеров в этих элементах указывают соответственно местоположение самого справочника и его порождающего.

Если номер начального кластера для порождающего справочника равен нулю, то порождающим является корневой справочник.

Когда размер файла уменьшается, ДОС возвращает освободившееся место на дискете в общий пул. Однако, если информация о файле хранится в справочнике нижнего уровня, то освободившееся место нельзя будет использовать, пока не будет уничтожен весь этот справочник.

5.7 Структура Таблицы Размещения Файлов (ТРФ)

Процедура распределения пространства памяти на дискете между файлами реализуется с помощью таблицы размещения файлов
(ТРФ). Использование ТРФ и интерпретация ее содержимого представляет собою непростую задачу - следовательно необходимо некоторое более подробное разъяснение. Основной принцип организации ТРФ заключается в создании таблицы, каждый элемент которой соответствует одному кластеру или фрагменту дискового пространства выделяемого файлу. Элементы таблицы содержат признаки занятости кластера или что он свободен и может быть отведен для файла. Доступные или

свободные элементы таблицы содержат нулевые значения. Участки пространства на дискете, принадлежащие файлу, связаны в цепочку. Элемент справочника файла содержит номер элемента в ТРФ, который соответствует первому из участков, выделенных для файла. Элемент ТРФ содержит номер элемента следующего файла и так далее, пока не будет достигут последний участок файла. В последнем элементе ТРФ для файла находится признак конца файла. Схема такой организации изображена на рис. 5.4 .

Такова в общих чертах схема ТРФ. Рассмотрим теперь ее подробнее.

Во-первых, рассмотрим кластеры выделяемого пространства памяти. Для односторонних дмскет пространство выделяется по одному сектору. Для двухсторонних дискет выделяются каждый раз два сектора . Если в кластер входит больше одного сектора, то объединяются последовательные сектора. Для двухсторонних дискет кластер состоит из четного и нечетного секторов одной и той же дорожки. Таким образом, каждая дорожка двухсторонней дискеты состоит из четырех кластеров на каждой стороне или всего восьми кластеров. В первый кластер входят сектора 1 и 2, во второй - 3 и 4 и так далее . (Если будут добавлены диски большей емкости, то можно будет организовать кластеры большего размера. В четвертой главе уже упоминалось, как можно динамически включить в BIOS программы обслуживания устройств для ДОС всех версий, начиная с версии 2.00. Описанная методика позволяет добавлять дисковые устройства, логическая организация которых будет совместима с используемыми дискетами. Помимо других характеристик, которые могут задаваться для новых устройств, можно указать и число секторов в кластере.)

Выделение места на двухсторонней дискете двухсекторными кластерами может показаться нерациональным, ведь в ряде случаев файл будет использовать лишь часть первого из двух секторов. Однако, пространство на двухсторонних дискетах теряется ничуть не больше чем на одностронних. Выделение пространства кластерами большими одного сектора позволяет ограничить увеличение размеров таблицы размещения файлов при использовании дискет все больших и больших размеров.

Раздел данных дискеты начинается сразу за справочником и занимает все место до конца дискеты. Первый кластер данных имеет номер 2 (почему это так, Вы сейчас поймете). На рис. 5.5 изображено расположение кластеров на одно и двухсторонних дискетах.

Как на односторонних, так и на двухсторонних дискетах хранится две копии ТРФ. Они находятся в секторах 2 и 3

нулевой дорожки нулевой стороны. Предполагается, что эти копии должны быть идентичны. Хранение двух экземпляров ТРФ представляет собой простую предосторожность, связанную с большой важностью информации, содержащейся в таблице. Восстановление запорченного справочника намного проще восстановления поврежденной ТРФ.

Полезность хранения двух копий ТРФ зависит от многих факторов, таких как степень возможного повреждения дискеты, степень совершенства программ или квалификация пользователей, выполняющих процедуры восстановления информации. Версии ДОС,
предшествовавшие версии 2.00, не включали специальных средств восстановления информации на дисках.

Как мне кажется, для ТРФ выделено два сектора не столько для того чтобы хранить две копии, сколько для обеспечения возможности увеличения этой таблицы в будущем. Она, конечно, не может увеличиваться до бесконечности, но два сектора позволяют увеличить ее объем вдвое.

Теперь можно подробно рассмотреть кодировку ТРФ. Простой подсчет показывает, что на дискете помещается больше 300 кластеров. Элемент ТРФ должен позволять хранить номер другого элемента ТРФ, а максимальный номер, который может храниться в одном байте - 255, следовательно, длина элемента ТРФ должна быть больше байта. С другой стороны, в 512-байтном секторе нельзя поместить больше 300 элементов ТРФ длиной по два байта. Для решения этой проблемы была разработана довольно сложная схема оформления элементов ТРФ в виде трех шестнадцатиричных цифр, т.е. длиной в полтора байта (12 разрядов).

Схема хранения чисел в виде полуторабайтных кодов выглядит довольно странно, хотя на машинном языке это реализуется очень просто.

Таблица при хранении сворачивается следующим образом: последовательные элементы ТРФ разбиваются на пары, обьединяющие два 1,5 байтовых значения, в последовательность из трех байтов для каждой пары /Здесь используется архитектурная особенность микропроцессоров фирмы "Интел", связанная с последовательностью хранения отдельных байтов в машинном слове (Прим.пер.)/. Для получения значения, хранящегося в элементе ТРФ с номером X, нужно выполнить следующие действия: сначала необходимо умножить X на 1.5 (для этого выполняется умножение на 3 с последующим делением на 2), затем полученное число используется в качестве смещения в ТРФ.

Двухбайтное число, хранящееся по указанному адресу загружается в регистр. Теперь в регистре находится четыре

шестнадцатиричных цифры, а необходимы только три из них. Если номер элемента ТРФ нечетный, то нужно отбросить последнюю цифру, а если он четный - то первую. Если описание Вам не совсем понятно,взгляните на рис. 5.6, который изображает процессы упаковки и распаковки пар элементов ТРФ, выполняемые вручную.

Рис.5.7 показывает связь между ТРФ в том виде, в каком она хранится на диске, и ТРФ в виде таблицы. ДОС переписывает ТРФ в память, но хранит ее в исходном, упакованном формате. Таблица ТРФ, показанная на рис.5.7, представляет собой логическую структуру ТРФ.

Чтобы увидеть ТРФ в исходном виде, воспользуйтесь программами DEBUG или DISKLOOK для считывания ТРФ с дискеты.
О программе DEBUG мы подробнее расскажем в главе 6, а пока приведем те команды, которые нужно ввести для считывания ТРФ:

```
L0011
DOL200
```

При работе с программой Disklook, нажмите функциональную клавишу F7 для выбора сектора, введите номер дорожки 0, номер сектора 2, после чего нажмите клавишу F6 для отображения всей таблицы.

Листинг программы 5.1 показывает определение на языке Паскаль той структуры, в которой ТРФ хранится на диске. Листинг 5.3 включает несколько программ для чтения, расшифровки и обслуживания ТРФ.

Вспомним, что нумерация кластеров начинается с 2. Это означает, что первые два элемента ТРФ (с номерами 0 и 1) не используются для хранения информации о размещении данных. Они
зарезервированы для хранения очень важной информации - сведений о формате дискеты. Код формата хранится в первом байте ТРФ. Соответствие кодов различным форматам приведено в следующей таблице:

ФОРМАТ	КОД ФОРМАТА
D - 8	FF
S - 8	FE
D - 9	FD
S - 9	FC
QD - 9	F9

Управляющие программы определяют формат дискеты путем чтения первого байта ТРФ. Программа, находящаяся в загрузочной записи, обращается к ТРФ еще и для того, чтобы определить где находятся два системных файла, IBMBIO.COM и IBMDOS.COM. С этим связаны различия в загрузочных программах ДОС версии 1.00 и версии 1.10.

Простые арифметические вычисления показывают, что на односторонней дискете находится 313 односекторных кластеров (со 2 по 314), а на двухсторонней дискете - 315 двухсекторных кластеров. Для односторонней дискеты такое число получается вычитанием из общего числа в 320 секторов 1 сектора загрузочной записи, 2 секторов ТРФ и 4 секторов справочника. Для двухсторонних дискет, из общего числа 640 секторов вычитается 1 сектор загрузочной записи, 2 сектора ТРФ и 7 секторов справочника, что дает 630 секторов или 315 двухсекторных кластеров. В одном 512-байтовом секторе можно разместить 340 элементов ТРФ, так что в конце секторов ТРФ остается немного свободного места.

В листинге программы 5.3 приведены процедуры на языке Паскаль, работающие с ТРФ. Эти процедуры могут быть использованы в больших программах. Их изучение поможет вам лучше разобраться в структуре ТРФ и в ее использовании.

Но это еще не все, что касается ТРФ. В этой таблице используются определенные коды специального назначения. Как мы уже видели, неиспользуемый кластер в ТРФ отмечается нулевым значением соответствующего элемента. Любое значение элемента таблицы в диапазоне от 2 до 314 (для односторонних дискет) или до 316 (для двухсторонних) является допустимым указателем на очередной элемент в цепочке, описывающей размещение файла. Восемь наибольших значений, которые могут храниться в элементе ТРФ, шестнадцатиричные числа от FF8 до FFF или десятичные от 4088 до 4095, используются для обозначения конца цепочки описания. Хотя необходим всего один код конца файла, резервируется восемь кодов на случай, если возникнет необходимость в каком-либо расширении. Восемь шестнадцатиричных кодов от FF0 до FF7 или десятичных от 4080 до 4087 оставлены для обозначения особых типов зарезервированных кластеров, например, кластеров с дефектными секторами.

На поверхности дискеты могут встречаться участки с дефектами магнитного покрытия, которые нельзя эффективно использовать для хранения данных. Такие области выявляются во время форматирования дискеты и исключаются из числа областей,

которые будут использоваться в дальнейшем. В первой версии ДОС 1.00, эти дефектные сектора оформлялись в виде скрытого файла с именем "badtrack.". Начиная с версии ДОС 1.10 используется более изящный метод.

Значение элемента ТРФ 4087 (или шестнадцатиричное FF7) используется для обозначения неиспользуемого кластера. Одно из достоинств такого метода пометки дефектных кластеров заключается в том, что не используется лишний элемент справочника. Программа DiskLook может быть использована для определения точного положения дефектных секторов на дискете, чего нельзя сделать стандартными средствами ДОС.

Любые другие значения элементов ТРФ, от 314 (или 316) до 4080, являются недопустимыми.

В организации ТРФ могут возникать определеннные дефекты. Два наиболее заметных дефекта - это "беспризорные" кластеры и перекрещивающиеся файлы. Если элемент ТРФ содержит значение,
указывающее, что элемент используется (значение не равно нулю) и, в то же время, этот элемент не входит ни в одну из цепочек определения размещения файлов, то такой элемент ТРФ становится как бы "беспризорным". Во втором случае, может случиться так, что две или больше различные цепочки, определяющие размещение файлов, приводят к одному и тому же кластеру. Такие файлы называются перекрещивающимися. Стандартная программа ДОС CHKDSK обнаруживает оба типа нарушений в структуре ТРФ и возвращает "беспризорные" кластеры в общий пул свободных и доступных для использования кластеров.

"Беспризорные" кластеры чаще всего возникают, когда программы начинают создавать файл (так что для него выделяются кластеры), но не закрывают его, вследствие чего не завершается создание элемента справочника для данного файла. Такое часто происходило с первой версией компилятора языка Паскаль в ДОС при обнаружении ошибки в исходной программе. Регулярное использование программы CHKDSK позволит избавляться от "беспризорных" кластеров, независимо от причин их появления. Нужно включать вызов CHKDSK в файл пакетной обработки с любой программой, в результате работы которой могут возникать "беспризорные" кластеры.

Если "беспризорные" кластеры встречаются достаточно часто, то перекрещивающиеся файлы возникают очень редко. Мне довелось их наблюдать только тогда, когда я специально их создавал. Если такая ситуация все же Вам встретится, нужно скопировать каждый из файлов на другую дискету для последующего восстановления (если оно потребуется). После этого, копии всех файлов будут содержать информацию из общих

секторов, хотя она может принадлежать только одному из них.

Нажав клавишу F9 при работе с программой Disklook Вы можете просмотреть карту диска с отмеченным на ней положением "беспризорных" кластеров и общих кластеров пересекающихся файлов.

При работе ДОС в памяти хранится копия ТРФ для каждого используемого дисковода. Когда в таблице производится какое-либо изменение, она записывается в обе копии на дискете. При новом обращении к дискете ДОС считывает ТРФ чтобы установить формат дискеты.

Теперь, познакомившись со структурами справочников и таблиц размещения файлов, можно рассмотреть форматы файлов данных. Сначала рассмотрим два наиболее распространенных формата: текстовые файлы (в кодах ASCII) и формат файлов с фиксированной длиной записи. Затем мы рассмотрим два формата хранения программ: ".COM" файлы и ".EXE" файлы.

5.8. Размещение файлов

Теперь можем рассмотреть метод выделения секторов данных для файлов. Этот метод довольно прост.

При записи на дискету данных файла для нее по одному выделяются кластеры секторов дискеты. Когда необходим очередной кластер для данных, то выбирается доступный кластер с наименьшим номером. Такая простая схема используется как при создании файла, так и при его удлинении (путем добавления данных в конец файла).

В отличие от некоторых других операционных систем (таких как р-система Калифорнийского университета в Сан-Диего), ДОС выделяет место на дискете по одному кластеру, когда в этом есть необходимость, не заботясь о том, чтобы все данные файла хранились в одной непрерывной области диска. При разработке любой схемы распределения дискового пространства в любой операционной системе приходится выбирать между свободным выделением пространства по одному кластеру, в результате чего файл может оказаться "размазанным" по всей дискете, и выделением места большими непрерывными сегментами , что усложняет задачу управления пространством на дискете. Операционная система ДОС использует более простой первый метод.

Если дискета пуста, то все доступное на ней место представляет собой одну большую чистую область. Когда файлы копируются на такую дискету, они оформляются в виде удобно размещенных непрерывных фрагментов дискового пространства. Но впоследствии, если файлы будут удлиняться, то дополнительное место будет выделяться в первых свободных кластерах, которые

могут находится в любом месте дискеты.

Когда файлы копируются на новую дискету и оставляются без изменения, их размещение на дискете остается экономичным. Но если создаются или удаляются какие-либо данные, то использование места на дискете становится весьма запутанным.

Это наиболее вероятно происходит, когда программа создает одновременно два файла. Если файлы увеличиваются параллельно, то их расположение на дискете обязательно будет перемежающимся.

Изучение динамики распределения пространства на дискете может оказаться довольно интересным и поучительным. Если у Вас есть моя утилита Disklook, то можно изучать распределение пространства на Вашей дискете, устанавливать расположение пространства всех файлов и проверять, фрагментировано ли пространство на диске.

Итак, схема распределения пространства на дискете в ДОС довольно хороша, она равномерно использует все пространство и не требует нашего вмешательства. Однако, все эти преимущества достигаются ценою "размазывания" файлов по дискете. Из-за этого может увеличиться время доступа к файлу, поскольку магнитная головка чтения/записи должна перемещаться по всей дискете для поиска нужных дорожек. Поскольку обращения к дискете являются наиболее медленными операциями и основными ограничивающими факторами производительности IBM/PC, такое размещение файлов может породить проблемы, хотя обычно этого не случается.

По этой причине, необходимо уделять внимание потенциальной проблеме фрагментации пространства, занимаемого файлами на дискете. Обычно эти проблемы Вас волновать не будут, но иногда они могут приобрести чрезвычайное значение. В тех случаях, когда фрагментация файлов покажется Вам нежелательной, скопируйте файл на новую дискету. Это же полезно делать, если Вам покажется, что дисковод слишком долго выполняет поиск дорожек при чтении файла.

При копировании файлов на новую дискету можно заодно улучшить их последовательность, разместив в начале часто использующиеся файлы или переставив их в таком порядке, в каком Вам хотелось бы их видеть в распечатке справочника. В некоторых случаях упорядочение данных может оказаться особенно полезным: файлы на дискете, хранящей корреспонденцию, лучше размещать в хронологическом порядке, для дискеты с множеством вспомогательных файлов предпочтительнее всего алфавитный порядок, ускоряющий просмотр. Полезно бывает, также, упорядочивать файлы по расширениям, которые обычно обозначают тип файла или способ его использования. Такое упорядочивание можно производить

либо вручную, ллибо с помощью какой-либо сервисной программы (такого рода программы имеются в составе пакета "Norton Utilities" (служебные программы Нортона)).

5.9.Формат текстового файла

Наиболее часто использующийся и один из самых важных форматов для файлов данных, размещаемых на дискетах,- это текстовый файл в коде ASCII. Такие файлы содержат обычную алфавитную информацию, тексты или отчеты, состоящие из алфавитных символов.

Текстовые файлы являются, вероятно, наибольшим приближением к универсальному формату для всех персональных компьютеров. Такой формат поддерживается практически всеми персональными компьютерами и имеет множество различных применений.

Большинство текстовых редакторов, включая и текстовые процессоры, используют текстовый формат. Редактор, поставляемый в составе ДОС, EDLIN, создает и использует текстовые файлы.

Текстовый формат используется для хранения исходного текста программ (исходной, нетранслировавшейся версии). Практически все языковые процессоры (такие как интерпретатор языка БЕЙСИК, компилятор языка Паскаль или Ассемблер) ожидают ввода входного файла в текстовом формате в коде ASCII.

Интерпретатор языка БЕЙСИК для IBM/PC работает с тремя форматами файлов программ на языыке БЕЙСИК одним из которых
является стандартный текстовый формат в коде ASCII. Для записи программы на языке БЕЙСИК в текстовый файл в коде ASCII нужно использовать ключ "A" в операторе SAVE:

```
10  SAVE "PROGRAM.BAS",A
```

Файлы пакетной обработки, одно из самых мощных и полезных вспомогательных средств ДОС, также хранятся в текстовом формате.

Чтобы понять структуру файлов, необходимо сначала понять саму кодировку, принятую в Американском стандартном коде для обмена информацией (ASCII). Код ASCII в основном состоит из кодов символов. Всем символам присваиваются определенные комбинации битов, например, код для заглавной буквы "A" будет выражаться шестнадцатиричным числом 41.

Существует 128 различных кодов в системе ASCII, поскольку в ней используются семибитовые коды. Большинство компьютеров, включая IBM/PC, используют восьмибитовую

кодировку символов, так что коды ASCII составляют лишь половину из ансамбля 256 возможных кодов. Для большинства компьютеров, и для IBM/PC, коды ASCII занимают первые 128 кодов, от CHR$ (0) до CHR$ (127).

Вообще говоря, имеется три различных категории кодов ASCII и две из них будут нас интересовать. Прежде всего, имеются коды ASCII, соответствующие буквам, знакам препинания, цифрам и т.п. Это первая из категории кодов ASCII и, конечно, они будут использоваться во всех наших работах, выполняемых с помощью персонального компьютера.

Следующую категорию я бы назвал символами форматирования. Использование этих символов следует уже из их названий, например, "возврат на шаг", "возврат каретки" и "перевод строки". Форматирующие символы разрабатывались специально для управления устройством печати. Эта категория нам также понадобится и мы еще к ней вернемся.

Третью категорию символов ASCII можно назвать символами связи. Они имеют такие имена как "начало-текста" или "конец-передачи" и используются главным образом в качестве управляющих сигналов при использовании кода ASCII для передачи информации по линиям связи. Эти символы позволяют управлять процессами установления связи и передачи данных. Полное разъяснение назначения этих символов тесно связано с изложением принципов осуществления взаимодействия по линиям связи, что не входит в сферу изложения данной книги. Поскольку эта категория символов не используется среди данных или в структуре файлов, она больше не будет нас интересовать.

Подведем некоторые итоги: одна из трех категорий кодов ASCII используется для кодировки данных, таких как буквы, цифры и знаки пунктуации. Вторая используется для форматирования, т.е., для указания того, где заканчивается одна строка и начинается другая; эти коды используются не только для управления форматом печати, но и для структурирования файлов. И, наконец, третья категория для управления передачей данных и никак не связаны с форматом текстовых файлов.

Обычные символы текста в коде ASCII имеют значения от CHR$(32) до CHR$(126). Символы ASCII двух специальных категорий - форматирующие и символы связи,- имеют коды от CHR$(0) до CHR$(31). Все эти коды должны использоваться для различных управляющих функций и они обычно используются отдельно от кодов данных. Если Ваша программа попытается непосредственно использовать эти коды, могут происходить странные вещи. Некоторые ситуации описываются в приложении 4.

Ниже приведены три простые программы на языке Бейсик, которые демонстрируют ряд экспериментов со специальными

кодами ASCII от 0 до 31:

```
10 REM вывод на экран дисплея
20 FOR I=0 TO 31
30 PRINT CHR$(I)
40 NEXT I
10 REM вывод на печать
20 FOR I=0 TO 31
30 LPRINT CHR$(I)
40 NEXT I
10 REM манипулирование режимом экрана
20 DEF SEG=&HB800   'цветная графика
30 DEF SEG=&HB000   'монохромный
40 FOR I=0 NJ 31
50    POKE 1*2,I
60 NEXT I
```

Теперь рассмотрим текстовые файлы в коде ASCII. Это файлы, содержащие текст, который состоит из обычных символов, например, букв алфавита. Символы текстового файла организуются в строки, подобно тому как они записываются на бумаге.

Границы строк отмечаются форматирующими символами кода ASCII: возврат каретки, CHR$(13), и перевод строки, CHR$(10). Можно было бы использовать любой символ для обозначения конца строки, но используются именно эти два символа, поскольку они управляют процессом перехода к новой строке на печатающем устройстве.

Одна из причин использования двух символов, обзначающих конец строки, состоит в обеспечении возможности наложения строк. Наиболее часто такая возможность используется для выполнения подчеркивания. При этом на печать в конце строки выдается только возврат каретки, а затем накладывающиеся символы. Поскольку перевод строки не выполнялся, новые символы печатаются поверх уже напечатанных. Такую операцию можно произвести с печатающим устройством, но нельзя с дисплеем.

Таким образом, физические команды, необходимые печатающему устройству для завершения строки, используются также в качестве логических меток конца строки в файле. Такой прием позволяет копировать текстовые файлы непосредственно на печатающее устройство, причем текст будет распечатываться правильно. Вы можете сами проверить это, использовав команду COPY для пересылки файла непосредственно на устройство печати или на зкран дисплея следующими командами:

COPY имя файла LPT1
COPY имя файла CON

Длина строки тектового файла ничем не ограничивается, строка продолжается, пока не встретятся символы возврата каретки и перевода строки. Однако, большинство программ, работающих с текстовыми файлами, ограничивают длину строки 255 символами, что обычно значительно больше, чем может потребоваться (некоторые редакторы и текстовые процессоры ограничивают длину строки данных таким образом, чтобы они укладывались в строку экрана дисплея: 80 символов).

Формат текстового файла определяется не только разделением на строки. Все файлы в коде ASCII имеют один общий элемент - маркер конца файла, значение которого CHR$(26). Этот маркер однозначно указывает конец файла. Обнаружение положения кода CHR$(26) позволяет точно установить размер файла, отличающийся от размера, указанного в справочнике.

С клавиатуры этот специальный символ с кодом CHR$(26) можно ввести, нажав клавишу "Z" и удерживая в нажатом состоянии клавишу "CONTROL" (управление). Если в любом тексте, связанном с персональными компьютерами, Вам встретится сочетание "CONTROL-Z", то имеется в виду функция конца файла, или значение CHR$(26) .

Кроме описанных трех универсальных кодов, возврата каретки, перевода строки и конца файла, имеются и другие форматирующие коды ASCII, которые используются с различной интенсивностью. Символ табуляции (CHR$(9)) используется для замены нескольких пробелов, следующих один за другим. Поскольку не существует общепринятого количества позиций табуляции, этот символ используется не так широко, как можно было бы ожидать.

Символ перевода страницы (CHR$(12)) обозначает переход к началу новой страницы. Большинство устройств печати отрабатывает этот код соответствующим образом и некоторые программы редактирования используют его для разделения текста на страницы. Использование этого символа также далеко от универсальности. Его стоит применять только для управления устройством печати.

Программы редактирования и, особенно, текстовые процессоры требуют большего числа символов форматирования и они определяют собственные коды для своих целей, таких как отметка границ параграфов, подчеркивания и тому подобного. Поскольку универсальных кодов для таких функций не существует, их включение в текстовой файл не позволит другим программам его использовать.

Прежде чем завершить рассмотрение текстовых файлов необходимо сделать еще ряд замечаний, касающихся обычных символов текста. Любая система кодирования определяет упорядоченную последовательность - своего рода эквивалент алфавитного порядка символов. В коде ASCII все обычные знаки пунктуации предшествуют буквенным кодам. Заглавные буквы располагаются перед строчными. Кроме того, в отличие от кода EBCDIC, используемого универсальными компьютерами фирмы "ИБМ", цифры располагаются перед буквами.

Коды заглавных букв отличаются от кодов строчных букв на 32 (на 32 меньше). Этот факт можно использовать в программах преобразования текстовых данных из верхнего регистра в нижний или наоборот.

5.10. Форматы записей данных

После текстовых файлов в коде ASCII наиболее распространены файлы с фиксированной длиной записей. В таком файле логические единицы информации называются записями и размер этих записей одинаков во всем файле. Файлы такого формата создаются интерпретатором БЕЙСИКА в качестве произвольных файлов, такой же формат имеют прямые файлы Паскаля и другие нетекстовые файлы. При выполнении каждой операции чтения или записи (обозначаемых в языке БЕЙСИК как INPUT# и WRITE#) пересылается одна запись.

Если текстовые файлы включают метки, отмечающие концы строк и конец файла, то в файле с фиксированой длиной записей, записи ничем не отделяются друг от друга. В таких файлах хранятся только данные без каких-либо разделителей.

Поскольку записи в таком файле имеют одинаковую длину, можно использовать простейшие арифметические вычисления для определения места, где заканчивается одна запись и начинается другая. Служебные программы ДОС для чтения и записи информации в файл, описанные в главе 4, используют такой метод как при последоваательном, так и при произвольном обращении к файлу. Когда Ваши программы обращаются к файлам, ДОС выполняет все функции поиска записей, так что Вашим программам не нужно заботится ни о каких подробностях, связанных с поиском.

Рассмотрим пример. Если длина записей файла 100 и у ДОС запрашивается чтение записи с номером 24, то ДОС вычисляет смещение в байтах от начала файла умножением 24 х 100. Разделив полученное значение 2400 на размер сектора, 512 байт, получим частноее 4 и остаток 352. Это значит, что запись расположена в пятом секторе файла (пропускаются 4

сектора) со смещением 352 байта.

(Такой способ подсчета предполагает, что нумерация записей начинается с нуля. Если номер первой записи единица, то необходимо сооветствующим образом изменить формулу, хотя сам метод остается неизменным).

В файлах с фиксированной длиной записи ДОС не приходится заниматься выравниванием записей - они располагаются одна за другой. Если длина записи не укладывается точно в 512-байтовый размер сектора, то некоторые записи могут размещаться в нескольких секторах, что несколько снижает эффективность операций чтения и записи.

В нашем примере со 100-байтными записями, очередная, 25 запись, частично размещается в 5-ом, а частично - в 6-ом секторе. Чтобы записать в файл эту запись, необходимо сначала считать, а затем записать обратно на диск два сектора.

Если Вы будете выбирать длину записи так, чтобы она была кратна 512, то скорость чтения и записи несколько повысится. ДОС, однако, позволяет работать с записями разной длины и, если не считать некоторого снижения эффективности при чтении/записи записей, располагающихся в нескольких секторах, Вам не нужно заботиться о размере записи.

Как Вы, вероятно, догадались, существует множество других форматов файлов, помимо текстовых файлов и файлов с фиксированной длиной записи. Мы так подробно остановились на этих двух форматах в связи с их широким распространением. Другие форматы хранения данных обычно имеют структурные требования, усложняющие процесс их чтения и записи, что может потребовать использования специальных программ для работы с ними. Существенное достоинство зтих двух форматов заключается в их гибкости - в самых различных ситуациях - так что они могут использоваться в очень многих приложениях. Поскольку они поддерживаются очень многими программами, в том числе различными языковыми процессорами и операционными системами,
их очень удобно использовать. На практике получается, что в связи с удобством работы с текстовым форматом и широкой поддержкой его различными программами редактирования и языками программирования, можно обнаружить случаи приспосабливания форматов данных программ к текстовому формату в коде ASCII.

5.11. Форматы программных файлов.

Существует два формата файлов для хранения программ. Для них зареэервированы два расширения имен: ".COM" и ".EXE". Подробности организации программных файлов потребуют полного

изложения организации ДОС, так что мы не будем сейчас их описывать, отметив только наиболее важные моменты. (Файлы программ БЕЙСИКА с расширением ".BAS" строго говоря не являются программными файлами - это файлы данных, которые считываются и используются интерпретатором БЕЙСИКА.)

Программные файлы типа COM с расширением имени файла ".COM" - это непосредственно загружаемые программы. В этих файлах хранится точный образ задач в том виде, в котором они находятся в памяти. Для них не требуется практически никакой подготовки к запуску после их загрузки в память средствами ДОС. Если Вы будете просматривать файл типа COM, например, программой DiskLook, Вы увидите команды машинного языка и данные, образующие программу, и ничего больше. Когда ДОС подготавливает програму из файла типа COM к выполнению, она формирует специальный сегмент-приставку, необходимый для всех программ, загружает программу, устанавливает регистры сегментов, присваивая им стандартные значения, и передает управление программе.

Программные файлы типа EXE с расширением имени файла ".EXE" несколько отличаются от файлов типа COM. Чтобы загрузить их для выполнения требуется определенная подготовка. Наиболее важная часть такой подготовки называется перемещением.

Программа может содержать адреса, значения которых должны меняться в зависимости от того, в каком месте памяти находится программа. Если программе это не требуется, то она называется само-перемещаемой. Программы типа COM должны быть
само-перемещаемыми, а перемещение программ типа EXE должно
выполняться специальной программой ДОС - загрузчиком.

Программы типа EXE начинаются специальной двухбайтовой меткой, 4D5A, которая используется для различения программ разных типов. За этой меткой следует определенная управляющая информация, включающая таблицу, указывающую какое перемещение
необходимо. Далее размещается собственно программа.

Помимо перемещения, во время загрузки для программ типа EXE может быть определен сегмент стека. Программы типа EXE могут определять куда они должны загружаться - в верхнюю или в нижнюю часть доступной памяти.

В связи с простотой формата COM, можно создавать относительно короткие и простые файлы типа COM в машинных кодах непосредственно, либо с помощью программ для модификации содержимого файлов на дискетах, таких как DEBUG или SecMod, либо путем написания короткой программы на

БЕЙСИКЕ, которая будет записывать байты программы прямо в файл типа COM. Такого рода программ создано уже очень много.

Формат типа COM проще, компактнее и быстрее загружается в память. Для программ, которым не нужны специальные услуги, обеспечиваемые форматом типа EXE, предпочтительнее использовать формат типа COM.

Приложение 5.1. Текст программы анализа структуры справочника (Паскаль).

```
const

  directory_sectors_single_sided=4;
  directory_sectors_double_sided=7;

  directory_entries_single_sided=64; {4*16}
  directory_entries_double_sided=112; {7*16}

  hidden_attribute          =wrd(2);
  system_attribute           =wrd(4);


{=================================================
==}

type

  directory_entry_type=

    record

      filename              : string(8);
      extension             : string(3);
      attribute             : byte; {hidden=2; system=4}
      reserved              : array[1..10] of byte;
      creation_time         : word; {hour*2048+minute*32+
                              secs}
      creation_date         : word; {(year-1980)*512+
                              month*32+day}
      starting_cluster_number  : word;
      file_size             : array[1..2] of word;

    end;
```

```
{=================================================
====}

complete_directory_type=array[1..112] of directory_entry_type;
```

Приложение 5.2. Текст программы анализа структуры таблицы размещения файлов (Паскаль).

```
const

  minimum_cluster               =    2;
  maximum_cluster_single_sided    =  314;
  maximum_cluster_double_sided    =  316;

  usual_fat_end_of_file          = 4095;
  minimum_fat_end_of_file        = 4088;

  bad_cluster_fat                = 4087;

  single_sided_indicator_byte     =  254;
  double_sided_indicator_byte     =  255;

{=================================================
====}


type

  fat_sector_type=array[0..511] of byte;

  fat_sector_structured_type=
    record
      scrambled_fat_pair: array[0..158]
of
        record
          scrambled_byte_1 : byte;
          scrambled_byte_2 : byte;
          scrambled_byte_3 : byte;
        end;
      unused_bytes_of_sector : array[1..35] of byte;
    end;
```

Глава 6. ВЫБОРКА ИЗ ПЗУ

[Оглавление]

Программы, размещаемые в пассивном запоминающем устройстве (ПЗУ) компьютера IBM/PC, определяют эффективность работы персонального компьютера в целом. В данной главе мы излагаем самые общие сведения, касающиеся ПЗУ. Мы рассмотрим
назначение ПЗУ, специфику анализа и декодирования (реконструкции) хранящейся там информации и, наконец, сравним между собой две первые версии ПЗУ компьютера IBM/PC. Все эти материалы являются базой для последующих пяти глав, в которых функциональные возможности IBM/PC рассматриваются более детально.

Здесь важно иметь в виду три аспекта. Все о чем пойдет речь в данной главе в полной мере относится только к IBM/PC и, вообще говоря, не распространяется на все семейство компьютеров, работающих под управлением MS-DOS. Для остальных
компьютеров семейства PC эти сведения следует рассматривать применительно к конкретному компьютеру (поскольку каждой модели присущи свои особенности). Тем не менее, имеются все основания считать, что прерывания или вызовы функции обслуживания для большинства компьютеров совпадают. Поэтому, несмотря на возможное изменение адресов программ обслуживания, находящихся в ПЗУ компьютера той или иной модели материалы настоящей и последующих пяти глав сохраняют свою актуальность.

6.1. Организация ПЗУ и его использование

Программы, хранящиеся в ПЗУ, представляют собой чрезвычайно важную компоненту IBM/PC, поскольку являются основой управляющей программы, делающей ПЭВМ эффективным
инструментом обработки данных. Сам факт их размещения в ПЗУ превращает эти программы в стабильный и надежный инструмент - они не могут быть стерты или уничтожены. Кроме того, они доступны для любой программы, выполняемой на IBM/PC, независимо от специфики используемой операционной системы.

Большинство программ обслуживания, размещаемых в ПЗУ, допускают перемещение. Эти программы написаны таким образом, что при необходимости их можно легко переместить,

соответственно перестроив обращение к ним. Специфика обращения к программам ПЗУ состоит в том, что их активизация производится всегда через систему прерываний. Изменив таблицу векторов прерываний, можно легко изменить точку входа в ту или иную программу обслуживания. Эти вопросы уже рассматривались нами при изучении материала главы 3.

Как уже говорилось выше, все пространство памяти ПЗУ компьютера IBM/PC делится на три части, под которые отводятся старшие адреса миллионного пространства памяти.

Самые старшие адреса памяти, соответствующие началу параграфа сегмента с адресом FE20/16, и охватывающие 8Кбайт до конца памяти отводятся под BIOS, или Базовую Систему Ввода-Вывода.Базовая Система Ввода-Вывода (BIOS) выполняет все операции по обслуживанию периферийных устройств IBM/PC, таких как дисплеи или накопители на гибких магнитных дисках. Все программы операционой системы, обеспечивающие функционирование ПЭВМ ориентируются на использование BIOS. Область памяти ПЗУ, содержащая BIOS, представляет особый интерес для тех, кто хотел бы в полной мере использовать возможности IBM/PC. В последующих пяти главах по мере рассмотрения конкретных функций IBM/PC мы будем детально обсуждать служебные обращения к BIOS, обеспечивающие реализацию этих функций.

Справочное руководство по техническому обслуживанию персонального компьютера содержит полный текст первой версии BIOS; при этом комментарии хорошо поясняют, что делается и как делается; вопрос, почему это делается, раскрыт слабо. Если Вы имеете навыки интерпретации листингов на языке ассемблера, то при изучении листинга BIOS Вы получите ответы на любые вопросы о принципах функционирования ПЭВМ.

Вторая часть ПЗУ, соответствует параграфу с шестнадцатеричным адресом F600 и занимает 32 Кбайт памяти. Она содержит (хранит) программы интерпретатора языка Бэйсик. В ПЗУ хранится ядро интерпретатора языка Бэйсик для IBM/PC. Все кассетные версии языка Бэйсик соответствуют Бэйсик-ПЗУ; основные функциональные возможности дисковой версии Бэйсика, а также расширенный Бэйсик (BASICA) фактически реализованы в ПЗУ. Практически все, что делают две указанные выше версии Бэйсика - это расширение версии Бэйсик-ПЗУ. Соответственно, Бэйсик-ПЗУ - это полная версия языка Бэйсик, содержащая основные программы, требуемые для дисковых версий.

С появлением в справочном руководстве по техническому обслуживанию текста программ BIOS не имет смысла приводить текст программ Бэйсик-ПЗУ; однако в ряде случаев мы будем использовать фрагменты этой программы для иллюстрации методов

исследования содержимого ПЗУ.

Сама по себе область ПЗУ, содержащая Бэйсик, не представляет особого интереса, однако коль скоро в этой области расположено множество полезных программ, то почему бы ими не воспользоваться в своей собственной разработке? Поиск и декодирование нужных фрагментов Бэйсика с последующим их тщательным документированием представляет собой весьма трудоемкую задачу, нами в данной работе не решаемую.

Подпрограмма или функция Бэйсика может быть непосредственно активизирована из программы пользователя с помощью механизма прерываний. В главе 11 эти вопросы рассматриваются с необходимой полнотой.

Третий и последний участок ПЗУ нами здесь не рассматривается. Фирма IBM оставляет за собой право расширять состав программ ПЗУ; пользователи также могут встраивать в ПЗУ свои программы, расширяя таким образом функциональные возможности IBM/PC. Дополнительные программы можно включать в
любое место миллионного адресного пространства, однако блок, начинающийся с шестнадчатиричного адреса параграфа F400 и занимающий 8Кбайт, вплотную прилегающий к участку памяти, отведенного под Бэйсик специально выделен для таких программ. Если требуется свыше 8Кбайт памяти, то программу можно сместить в область памяти с более низкими адресами. Именно посредством включения в ПЗУ специализированных программ можно
превратить IBM/PC в спкциализированный компьютер. Прежде чем приступить к более глубокому изучению BIOS мы подемонстрируем (в двух последующих параграфах) способы исследования содержимого ПЗУ.

6.2. Анализ содержимого ПЗУ средствами программы DEBUG

В данной главе мы рассмотрим особенности анализа программ и данных, размещаемых в ПЗУ. Несмотря на то, что предлагаемая методика ориентируется главным образом на программы, расположенные в ПЗУ, она может использоваться также и в случае программ, хранящихся на дискете. Поэтому все, о чем пойдет здесь речь применимо как к компонентам DOS, расположенным на дискете, так и к компонентам, входящим в состав других программ.

Доступ к любому фрагменту ПЗУ может быть осуществлен также средствами Бэйсика или Паскаля. Методы адресации объектов памяти, рассмотренные в главе 3, позволяют работая

на Бэйсике или Паскале, обратиться к любой ячейке памяти. Ниже приводится пример программы на Бэйсике, отображающий содержимое ПЗУ.

```
100 REM     отображение фрагмента ПЗУ в шестн. формате
110 PARAGRAPH=&HF600
120 DEF SEG=PARAGRAPH
130 PRINT
140          PRINT          "Отображение          содержимого
параграфа",HEX$(PARAGRAPH)
150 PRINT
160 PRINT "Десятичное       Шестнадцатеричное"
170 PRINT "Смещение   Смещение   Содержимое ПЗУ";
180 OFFSET%=0
190   FOR LINES%=1 TO 16
200     PRINT
210     PRINT OFFSET%, HEX$(OFFSET%)
```

Хотя приведенная программа позволяет нам судить о содержимом ПЗУ, смысл даных остается "за кадром". Мы, таким образом, подошли к очень важному и необходимому инструменту, поставленному в составе DOS - программе DEBUG.

Программа DEBUG позволяет осуществлять три вида действий вотношении содержимого ПЗУ: выборку произвольного участка памяти и отображать ее содержимое в двух форматах:

1)шестнадцатеричный / ASCII.

2)формат не связанный с деассемблированием.

Программа DEBUG может быть также использована для исследования программ, хранящихся на дискете. В последнем случае, правда, представляется более эффективным использовать программу Disklook, записанную на дискету, прилагаемую к данной книге или программу SecMod входящую в состав программ Norton Utility. Прежде чем двигаться дальше, поясним смысл термина "деассемблирование".

Программы, представленные на языке машинных команд, чрезвычайно сложны для восприятия человеком. Даже в случае компьютеров, имеющих сравнительно простой формат машинных команд, этот язык весьма сложен для понимания. Микропроцессор 8088 регламенитрует очень сложную структуру языка машинной команды. Это положение справедливо в отношении большинства микропроцессоров - их проектировщики жертвуют простотой для достижения максимальной производительности и компактности кода. Вследствие этого, требуется приложить значительные усилия для просмотра и понимания листинга, представленного на языке машинных команд в шестнадцатеричном коде.

Процесс деассемблирования, реализуемый программой DEBUG,

значительно облегчает поцедуру интерпретации машинного языка. Деассемблирование - это процесс трансляции или преобразования инструкций машинного языка, представленных в абсолютном шестнадцатеричном виде в символическую нотацию языка ассемблера. Так, например, на языке ассемблера можно записать INC AX (увеличить содержимое регистра AX на единицу); ассемблер переведет эту конструкцию в команду на машинном языке с кодом 40/16. Функция деассемблера состоит в том, чтобы привести команду машинного языка с кодом 40/16 обратно к виду INC AX.

Шестнадцатеричный код 40 сравнительно легко может быть преобразован и вручную, при наличии таблицы кодов команд микропроцессора 8088. С большинством кодов остальных команд дело обстоит не так просто, поскольку в ряде случаев смысл команды изменяется в зависимости от значения кокнретных битов. В случае использования команды DEBUG все запутанные случаи декодирования преодолеваются в режимах команды деассемблирования.

Несмотря на то, что деассемблер может выполнить преобразование команд машинного языка к более удобным командам языка ассемблера, целый ряд весьма существенных компонентов программы, написанной на языке ассемблера реконструкции не поддается. Так, очевидно, не могут быть восстановлены весьма полезные комментарии программиста. Кроме того, не представляется возможным восстановить оригинальные символические имена адресов памяти.

Таким образом, вместо оригинальной авторской конструкции типа

IMP FINISH; выполнено, перейти к завершающей процедуре

деассемблер сгенерирует строку вида

IMP OE6C

Пользователь видит перед собой команду перехода, однако ее смысл остается для него неясным.

Деассемблер, безусловно, не в состоянии полностью заменить программиста, особенно, в том, что касается смысла деассемблируемой программы. Но он существенно упрощает задачу, когда предъявляет пользователю команду, а также помогает отличить область памяти, содержащую данные от области памяти, содержащую команды.

Для иллюстрации особенностей исследования программ ПЗУ возьмем программу Бэйсика и используем ее в качестве объекта приложения программы DEBUG. Все дополнительные сведения будут
сообщены ниже, либо проясняться в ходе рассмотрения примера. Для детального изучения программы DEBUG - следует внимательно

прочесть сответствующие разделы руководства по операционной системе; этот материал не прост для усвоения, однако он содержит многие важные технические детали.

Для пользователей, не работавших с программой (командой) DEBUG, сообщим некоторые дополнительные сведения. Подобно остальным командам, эта команда запускается путем набора на клавиатуре ее имени. В процессе ее выполнения запосам на ввод исполнительных операторов предшествуют лишь "тире": обычный запрос операционной системы (символ "A") в этом случае не используется.

Большинство действий с программой DEBUG отображаются на экране дисплея, однако может возникнуть потребность в их распечатке для последующей работы. Собственно программа DEBUG
не выводит результаты своей работы на принтер.

В этой связи, однако, следует напомнить, что существует два способа получить твердую копию. Первый способ заключается в использовании для копирования текущего кадра клавиши PrtSc (Печатать содержимое экрана); второй - состоит в использовании принтера для эхо-отображения путем одновременного нажатия клавиш Ctrl и PrtSc.

Итак, мы приняли решение об исследовании участка ПЗУ, содержащего Бэйсик; нам также известно, что адрес начала этой программы соответствует шестнадцатеричному адресу F600. Можно запускать программу DEBUG и приступать к непосредственной работе с этой программой.

Предположим теперь, что адрес объекта наших поисков (F600) нам неизвестен. В этом случае поиск может вестись в двух направлениях. Первое - это методичное использование операции D (означающее "дамп" или "отображение") до тех пор, пока не появится что-нибудь интересное. Второе - более эффективное направление - состоит в предъявлении программе DEBUG образца объекта, который, по нашему мнению должен находиться в памяти.

В качестве образца объекта поиска выберем сообщение об ошибке, выдаваемое Бэйсиком: "Illegal Function call" (Неверный вызов функции). На рис 6.1 показана процедура запуска программы DEBUG и оператор поиска указанного выше сообщения программы Бэйсик. Предположим, для определенности, что в качестве начального адреса просмотра (поиска) программе DEBUG был передан адрес параграфа F000 (лежащий немного ниже
адреса начала Бэйсика), а также размер (длина) просматриваемого учестка (L) равный 65535 байтам (`FFFF'в шестнадцатеричной системе счисления). Программа DEBUG сообщает, что интересующий нас текстовой объект найден. Это

сообщение формируется в форме сегментированного адреса F000: 63F4, означающего, что относительно параграфа F000 берется шестнадцатеричное смещение 63F4. Выполнив действия над адресами, мы получим полный 20-байтовый адрес.

```
   F000
+
   63F4
_____________
   F63F4
```

```
A > DEBUG

- S F000 : 0000 L FFFF "Illegal Function call"

F000 : 63F4
```

Рис. 6.1. Запуск программы DEBUG и поиск сообщения

Значение адреса искомого текстового объкта говорит о том, что найденное сообщение расположено недалеко от начального параграфа размещения Бэйсика F600. Если представить 20-ти битовый адрес F63F4 в сегментированном виде, используя в качестве базы параграф F600, то будем иметь адрес F600:03F4. Таким образом, относительное смещение от начала Бэйсика составляет только 3F4/16 или 1012/16 байтов.

Вводя команду D F600 : 03E0 (вывод на экран содержимого области памяти, начиная с адреса - т.е. области памяти, начинающейся несколько раньше найденного сообщения - относительно базового адреса параграфа F600) можно отобразить содержимое близлежащего пространства памяти, включающего область найденного сообщения. На рисунке 6.2. представлен результат выполнения этой команды. Программа DEBUG выводит на
устройство отображения - в шестнадцатеричном и символьном (ASCII) форматах - содержимое участка памяти длиной 80 байт, причем этот участок памяти содержит искомое сообщение. Нам таким образом представлено искомое сообщение, а также целый ряд других сообщений об ошибках Бэйсика, т.е. именно то, что - как предполагалось - должно находиться где-то поблизости. При необходимости с помощью программы DEBUG можно продолжить
отображение на экране последующих участков памяти; для этого следует ввести команду D, не указывая адреса начала ячейки.

Рисунок 6.2. иллюстрирует ряд возможных особенностей, которые следует иметь в виду, используя команду "D" в рамках программы DEBUG. Одна из этих особенностей состоит в том, что в левую часть поля вывода помещается шестнадцатеричная информация, а в правую - информация в коде ASCII. Вторая особенность заключается в том, что кодовая комбинация не имеющая символьного представления в коде ASCII изображается в правой части поля вывода с помощью точки. Отсюда, в частности, вытекает важное следствие: для того, чтобы использовать эхо-печать программа DEBUG контролирует поступающие на ее вход кодовые комбинации, не имеющие представления в коде ASCII; эти комбинации могут быть восприняты принтером в качестве управляющего кода. Вследствие этого, нельзя расчитывать, что правая часть поля вывода (информация в коде ASCII) будет давать объективную картину. Все, что может быть приведено к стандартным печатным символам, или является печатным символом (со старшим битом в единичном состоянии) будет отпечатано, все остальное будет преобразовано в точки.(Здесь уместно заметить, что программа DiskLook, входящая в состав пакета программ, записанного на дискету, прилагаемую к настоящей книге, отображает в точности все шестнадцатеричные коды и коды ASCII, поскольку выводит информацию только на экран и не использует эхо-печать).

До сих пор мы рассматривали способы отображения данных, или поиска априори известных данных, хранящихся в ПЗУ с помощью программы DEBUG. Ниже мы будем решать существенно
более сложную задачу - задачу деассемблирования и интерпретации программного кода на машинном языке.

6.3. Анализ содержимого ПЗУ - метод деассемблирования

Наш следующий шаг в решении задачи исследования содержимого ПЗУ и использования программы DEBUG состоит в знакомстве с особенностями процесса деассемблирования. Команда "U" программы DEBUG - означает "деассемблирование" - осуществляет преобразование произвольных кодов памяти в мнемонические коды языка ассемблера. Деассемблер реализует: перевод абсолютных шестнадцатеричных кодов программ, в коды команд на языке ассемблера (например: AX или DS). Деассемблер не делает две вещи.

Первое. Деассемблер не интерпретирует смысл программы и не обучает пользователя. Для понимания листинга, выдаваемого деассемблером, необходимо либо знать язык ассемблера, либо

иметь желание покопаться в справочном руководстве и, установив смысл мнемонических обозначений, постараться понять суть дела. (Для этой цели могут быть использованы следующие источники: "Руководство по языку ассемблера фирмы IBM", справочники по системе команд микропроцессора 8086/8088, такие как "Учебное пособие по микропроцессору 8086" С.Морса или "Описание микропроцессора 8086" Ректора и Алекси).

Для понимания программы на языке ассемблера (или программы, приведенной к языку ассемблера), знания одного языка Ассемблер недостаточно, необходимо также иметь некоторое представление о функциональной ориентации программы.

Постижение пограммы представляет собой занимательное умственое упражнение, своего рода головоломку. Со временем эта процедура становится все проще и проще, отчасти потому, что времени на поиск смыслового описания команды уходит все меньше и меньше, а отчасти потому, что усваиваются приемы программирования на ассемблере и способы достижения целей. Далее мы остановимся на этих вопросах более подробно.

Второе, что не под силу программе деассемблера - это установка так называемой абсолютной синхронизации. Известно, что команды машинного языка для микропроцессора INTEL 8086/8088 имеют переменную длину - от одного до шести байтов. После того как деассемблеру сообщена конкретная позиция памяти он приступает к процедуре прямого декодирования, не отличая кодов команд от данных. Достаточно ошибиться в выборе исходной позиции памяти (например, попасть не на границу между командами или в область данных) и результат окажется неверным.

Если начальная точка набора команд известна, то никаких проблем не возникает. Если какие-либо предварительные сведения отсутствуют, то следует провести необходимое исследование. Первое, что необходимо сделать - это выбрать исходную точку в наборе команд и запустить операцию деассемблирования. После этого следует предпринять попытку интерпретировать результат работы деассемблера. Если осмысленные программные участки отсутствуют (ниже мы приводим некоторые соображения по их выявлению),то следует повторить процедуру, сместив начальную точку на один, два или три байта по отношению к предыдущей попытке. Мы таким образом пытаемся отыскать точку синхронизации, расположенную на стыке двух команд, после чего может быть восстановлен оригинальный набор команд. Если очередная попытка закончилась неудачно, то следует попробовать еще раз.

Поскольку большинство команд имеют длину один или два байта, то точку синхронизации отыскать как правило несложно.

Кроме того, необходимо учитывать, что процессу деассемблирования присущ элемент самосинхронизации. Даже если
в качестве начальной точки для деассемблирования выбрана точка, расположенная внутри команды, то часто случается, что процесс деассемблирования при обработке последующих команд самопроизвольно попадает в нужную точку. После этого все идет гладко, по крайней мере в сторону увеличения адресов. (В обратную сторону это приходится делать вручную).

Выполняя деассемблирование программы необходимо внимательно следить за данными (это нечто противоположное командам), которые деассемблер превращает в причудливые ассемблерные конструкции. Основной и наиболее быстрый способ обнаружения данных состоит в использовании команды "D", для представления информации в шестнадцатеричном коде и коде ASCII. Здесь в первую очередь следует обращать внимание на самые очевидные вещи - содержательно осмысленную последовательность символов ASCII (фразы), такую, например, как "неверный вызов функции". После этого можно приступать к выявлению более тонких признаков данных.

В обычных программах (т.е. в программах, размещаемых вне ПЗУ) наличие полей, заполненных шестнадцатеричными нулями является признаком рабочей области данных, то есть такой области, которая будет использоваться в ходе выполнения программы; в процессе деассемблирования данные этой области неактуальны. ПЗУ не может использоваться для хранения рабочих данных, поскольку запись в эту память невозможна. Таким образом, наличие полей, состоящих из шестнадцатеричных нулей нетипично для ПЗУ-программ.

Признаком возможного наличия данных являются байты или двухбайтовые слова, для которых второй или старший байт содержит 0 или F. Дело в том, что значительное число констант программ являются малыми числами, положительными (т.е. начинающихся с 0), либо отрицательными (т.е. начинающихся с F). Если будет обнаружена последовательность байтов, удовлетворяющая этим требованиям, то весьма вероятно, что найдена область памяти, используемая для хранения одного или двухбайтовых чисел. Эта часть программы не может содержать команды.

Пытаясь осмыслить структуру листинга, выдаваемого деассемблером, необходимо иметь в виду следующие положения. Обычные программы, прошедшие этап редактирования связей, будут хранить команды в одном месте, а данные в другом. Такой подход является элементом хорошего стиля программирования; именно так работает редактор связей дисковой операционной системы. Описанный принцип характерен для программ всех

типов, однако для программ типа EXE он справедлив в большей степени чем для программ типа COM. С другой стороны, ПЗУ-программы, подобные тем, которые рассматривались нами выше, часто разрабатываются вне концепций технологии программирования. Данные в этих программах могут следовать вперемешку с командами.

Если предположительно обнаружен участок программы, в котором вслед за командами следуют некоторые данные, то существует по крайней мере один достоверный способ отличить конец последовательности команд от расположенных вслед за ними данными, и таким образом избежать необходимости рассматривать недостоверный протокол деассемблирования. Последняя команда фактического участка программы должна представлять собой ту или иную разновидность команды перехода или ветвления. Команды ветвления включают любые команды переходов, кроме команд условных переходов. В качестве общеупотребительной команды завершения программы принято использовать команду возврата (RET) (возврат из вызванной ранее подпрограммы).

Могут использоваться также и команды CALL (Вызов) и INT (Прерывание); хотя их использование в качестве команды завершения последовательности команд встречается достаточно редко.

Существует ряд особенностей, на которые следует обращать внимание при решении вопроса подлинности предъявленного участка программы после его деассемблирования. В первую очередь следует обратить внимание на используемые регистры. Если операции реализуются на регистрах, используемых как правило для выполнения арифметических операций (AX или AL, AH;BX или BL, BH; CX или CL, CH; DX или DL, DH), но никакие действия в отношении результата не предпринимаются, то это вызывает подозрение. Здесь следует быть более внимательным. Один из результатов выполнения арифметических операций состоит в установке флагов - следовательно, естественно ожидать наличие команд условных переходов по состоянию флага, таких как JNC (команда перехода по условию отсутствия установленных битов флага).

Рассмотрим аспект некорректности использования регистров в реконструируемой программе. Программы не используют регистры некоторым хаотичным образом - им присуща определенная дисциплина. Поэтому весьма непривычно, скажем, видеть загрузку или непосредственную ссылку на сегментные регистры - особенно сегментный регистр программы (CS) и сегментный регистр стека памяти (SS). Загрузка сегмента регистра данных (DS) более вероятна, но также встречается редко. С другой стороны, дополнительный сегментный регистр

(ES) практически постоянно используется программами, поэтому его загрузка - дело обычное. Если загрузка регистра CS или SS все-таки обнаружена, то это должно быть сделано в программной секции, формирующей операционную структуру выполняемой программы; в этом случае вероятнее всего, что в одном месте будет произведена загрузка нескольких сегментных регистров, в частности, CS, SS и DS.

Попытка реконструировать программу по листингу, выданному деассемблером или даже просто попытка убедиться в подлинности предъявленного набора команд, представляет собой захватывающее умственное упражнение. Для этого,кстати, не требуется специальная подготовка. Как правило, достаточно бывает просто здравого смысла. Усвоив все наши рекомендации и приобретя некоторые практические навыки, Вы, при необходимости, сможете все это проделать. В следующем параграфе мы выполним декодирование (реконструкцию) программы
Бэйсик-ПЗУ; читатель, таким образом, получит представление о том, как это делается и убедится в том, что это не очень сложно.

6.4. Анализ содержимого ПЗУ - реконструкция интерпретатора

В качестве примера попробуем деассемблировать фрагмент Бэйсик-ПЗУ. Начнем с исходного параграфа, т.е. параграфа с адресом F600(16). На рисунке 6.3 представлен результат деассемблирования первых 32 байтов. Шестнадцатеричные нули в конце фрагмента программы (преобразованы деассемблером в команды ADD (Сложить)) представляют собой данные. Они "хорошо" иллюстрируют высказанное ранее положение о том, что нулевые данные нетипичны для ПЗУ-программ. Команда DB (определить байт), предшествующая полю нулей, появилась как реакция деассемблера на кодовую комбинацию, которая не может быть преобразована в команду ассемблера (такая ситуация, благодаря широкому набору команд, встречается довольно редко). Итак, являются ли все данные, которым предшествует команда безусловного перехода собственно осмысленными командами? Предварительный ответ на этот вопрос может быть положительным, поскольку для завершения выполнения подпрограммы используется команда возврата (RET). Пока все идет хорошо.

Можно ли приписать какие-то осмысленные действия первым пяти командам деассемблированной программы? Эти команды

выглядят несколько странно, поскольку начинаются с команды безусловного перехода (JMP). Большинство программ, однако, первым делом выполняют ряд действий настроечного плана, обращаясь для этого посредством команд вызова подпрограмм или переходов, к нужным процедурам. Таким образом, безусловный переход в начале программы - каким бы причудливым он не казался - не такая уж бессмыслица. Следующие четыре команды - две пары обращения к подпрограммам с последующими командами
возврата в процедуру не особенно понятны, но, по крайней мере, не противоречат друг другу и могут в какой-то степени служить признаком наличия последовательности команд. Можно, следовательно, сделать предположение о том, что мы имеем дело с реальной последовательностью команд.

Из того, чем мы располагаем - даже при том, что на рисунке 6.3 представлена реальная последовательность команд - вряд ли можно извлечь какую-то пользу. Только пять команд почти ничего не совершающих.

И все-таки есть ряд интересных моментов. Если это действительно реальная последовательность, то нам названы три адреса, где могут оказаться вещи более интересные. Речь идет об адресе в команде перехода и адресах двух последующих каманд обращения к процедуре. Следует ли нам пойти по адресу перехода? Деассемблер дает нам этот адрес (относительно начала нашей программы, т.е. адреса F600/16) - FE92/16.

Выполним деассемблирование небольшого участка памяти, начинающегося с этого адреса (результат приведен на рис 6.4.)

```
-U F600:0000
F600:0000 E98F7E        JMP         7E92
F600:0003 E8A76B        CALL        6BAD
F600:0006 CB            RET         L
F600:0007 E80265        CALL        650C
F600:000A CB            RET         L
F600:000B C1            DB          C1
F600:000C 0000          ADD         [BX+SI],AL
F600:000E 0000          ADD         [BX+SI],AL
F600:0010 0000          ADD         [BX+SI],AL
F600:0012 0000          ADD         [BX+SI],AL
F600:0014 0000          ADD         [BX+SI],AL
F600:0016 0000          ADD         [BX+SI],AL
F600:0018 0000          ADD         [BX+SI],AL
F600:001A 0000          ADD         [BX+SI],AL
F600:001C 0000          ADD         [BX+SI],AL
```

```
F600:001E 0000          ADD         [BX+SI],AL
```

Рис. 6.3. Деассемблирование начального участка Бэйсика

Похоже, мы попали в точку. Во-первых, перед нами совокупность осмысленных команд, не содержащая ни последовательностей операторов DB, ни последовательностей операторов ADD.Во-вторых, эти команды весьма напоминают команды настройки.Об этом говорит прежде всего команда CLI, (CLI (Clear interrupt flag) - очистка флага прерывания).

Прерывание в микропроцессоре INTEL 8086/8088 воспринимается по завершении выполнения некоторой команды. Немаскируемое прерывание в общем случае обслуживается непосредственно после выполнения текущей команды. Прерывание по вектору воспринимается только тогда, когда разряд IF (флаг прерывания) в регистре FLAGS имеет значение, равное 1. Для того, чтобы микропроцессор 8086 воспринял прерывание, необходимо выполнение следующих трех условий:

1) разряд IF регистра FLAGS должен иметь значение, равное 1 (для прерываний по вектору);

2) разряд разрешения прерываний в интерфейсе устройства должен находиться в состоянии "Прерывание разрешено";

3) интерфейс должен зафиксировать некоторое событие, вызывающее прерывание (например, поступление символа с клавиатуры, готовность к выводу символа на дисплей, завершение передачи блока данных с диска и т.п.) (Прим.перев.)).

которая, как мы видели в главе 3 используется как средство блокировки прерываний; таким образом вся последовательность выполнения расположенных ниже команды прервана быть не может.

Именно такая последовательномть команд характерна для начальных аргументов ответственных программ, поскольку значения сегментных регистров целесообразно устанавливать одновременно с обработкой прерываний.

Ниже производятся как раз те действия, о которых мы говорили выше, а именно: загрузка сегментных регистров. С помощью четырех команд MOV (команды пересылки данных) осуществляется загрузка трех сегментных регистров из четырех: DS, ES и SS. Загрузка этих регистров осуществляется весьма редко, поэтому естественно это сделать один раз в начале программы. (Сегментный регистр программы CS к этому моменту уже загружен (F600 : 0000, рис. 6.3.).

Вслед за группой, состоящей из четырех команд MOV,

осуществляющих загрузку сегментных регистров, следует две команды, выполняющие одну логическую операцию. Первая команда
- это команда "исключающего ИЛИ" XOR - заносит в регистр AL значение "нуль" (поскольку в результате применения "исключающего ИЛИ" к эквивалентным данным образуется новая величина), а вторая команда заносит эту величину по конкретному адресу.

Программа не содержит никаких сведений относительно содержательной интепретации этих операций, но это вполне осмысленное сочетание команд, выполняющих конкретную функцию.

Если бы мы располагали более полной информацией об особенностях работы Бэйсика, то нам, вероятно, было бы известно, что нулевая величина, пересылаемая в память с помощью двух двух последовательных команд представляет собой переключатель предохраняющий защищенные программы. При загрузке защищенных программ, написанных на языке Бэйсик, этот переключатель устанавливается с целью предотвращения выдачи их на печать. Это, однако, никак не следует из представленного листинга.

Первые семь команд рассматриваемого участка программы, таким образом, выглядят вполне осмысленно. Они реализуют рациональные и согласованные между собой действия и естественным образом распадаются на два класса: команды со 2-й по 5-ю реализуют одну логическую операцию, а с 6-й по 7-ю - другую. Пред'явленные команды отвечают всем критериям реального участка программы.

```
-U F600:7E92
F600:7E93 8A6000          CLI
F600:7E93 8A6000          MOV      DX,0060
F600:7E96 8EDA            MOV      DS,DX
F600:7E98 8EC2            MOV      ES,DX
F600:7E9A 8ED2            MOV      SS,DX
F600:7E9C 32CO            XOR      AL,AL
F600:7E9E A26404          MOV      [0464],AL
F600:7EA1 B591            MOV      CH,91
F600:7EA3 BB0000          MOV      BX,0000
F600:7EA6 BA9A06          MOV      DX,069A
F600:7EA9 BBF2            MOV      SI,DX
F600:7EAB 2E              SEG      CG
F600:7EAC AC              LODSB
F600:7EAD 8807            MOV      [BX],AL
F600:7EAF 43              INC      BX
```

```
F600:7EBO 42              INC      DX
F600:7EB1 FECD            DEC      CH
```

Рис. 6.4. Второй фрагмент реконструированного Бэйсика

Мы в какой-то мере достигли относительной цели: поиск и выявление некоторого программного кода, а также некоторое его осмысление. Это нам вполне удалось. Можно было бы выполнить декодирование всего Бэйсика, либо наиболее существенных его участков. Однако мы уже получили представление о том, как это удается и убедились, что это вполне возможно.

6.5. Две версии BIOS'a

То, что для IBM/PC используется по крайней мере две различные версии BIOS не является широко известным фактом. В настоящем параграфе мы кратко рассмотрим имеющиеся между нами различия и оценим их применительно к своим нуждам.

В состав пакета программ, записанного на дискету и прилагаемого к настоящей книге, входят две программы копирующие и анализирующие программы BIOS.

Именно с помощью этих программ впервые было обнаружено наличие модифицированных версий ПЗУ компьютеров, факт,который открыто не признавался фирмой IBM. С помощью этих программ можно проверить свою систему и сопоставить свою версию с другими.

Из всех программ, используемых для Вашего компьютера IBM/PC, наиболее совершенными (не содержащими ошибок) должны быть программы, предназначенные для размещения в ПЗУ. Если, например, будет выявлена ошибка в какой-либо версии DOS или VisiCalc, то ее можно устранить путем простой перезаписи дискеты. С программами, записанными в ПЗУ, дело обстоит гораздо сложнее. Их можно записывать только путем замены корпусов микросхем памяти, расположенных на главной плате компьютера IBM/PC; эта операция не может быть произведена силами обычного пользователя. Наличие ошибок в программах ПЗУ представляет собой серьезную проблему для фирмы IBM.

Сама мысль о том, что можно сделать что-либо стоящее, не совершив при этом ни единой ошибки, представляется

фантастической и программы ПЗУ в этом смысле не исключение. С момента появления компьютеров IBM/PC, фирма IBM сочла необходимым внести ряд изменений в программы ПЗУ (речь идет о серьезных ошибках), что обусловило появление двух версий ПЗУ. У пользователя может возникнуть вопрос, какой версией оснащен его компьютер и чем эта версия отличается от другой версии.

В самом конце программы BIOS фирма IBM поместила временную отметку, так называемый маркер версии. Этот маркер практически соответствует дате официального завершения разработки программ ПЗУ. Для его отображения на экране можно воспользоваться программой DEBUG. После загрузки программы DEBUG введите команду D F000:FFF5 L 8.

На экране вашего компьютера будет отображена последовательность символов, обозначающая маркер версии: 04/24/81. Маркер второй версии имеет вид 10/19/81. Если будет обнаружен маркер версии, отличной от этих двух, это будет означать, что Вы располагаете версией BIOS-ПЗУ, отличной от указанных выше.

В приложении 6.1 приведен текст программы, проверяющей маркер версии BIOS-ПЗУ, и, в зависимости от его значения, выполняющей те или иные действия. Аналогичные действия на БэйсикЕ можно представить следующим образом:

```
10 REM    вывод на дисплей маркера версии ПЗУ
20 REM    программа отображает маркер, не анализируя его
30 REM    и не настраивается на версию ПЗУ
40 PRINT
50 DEF SEG=&HFFFF
60 PRINT "Маркер версии ПЗУ;
70 FOR OFFSET=5 то 12
80 PRINT CHR$(PEEK(OFFSET));
90 NEXT OFFSET
```

Если Ваш компьютер приобретен позже октября 1981, то это еще не означает, что он снабжен последней версией BIOS-ПЗУ. Проверка компьютеров, выпущенных через год после выпуска второй версии BIOS, показывает, что новый, только что собранный компьютер, может иметь первую версию BIOS-ПЗУ.

Вы, вероятно, помните, что ПЗУ IBM/PC содержит две составляющие: Бэйсик-ПЗУ и BIOS-ПЗУ. Первое, что было обнаружено с помощью разработанной автором программы, анализа содержимого ПЗУ, это отсутствие различий между первой и второй версиями Бэйсик-ПЗУ.

Все ошибки, обнаруженные в БейсикЕ, были устранены только за счет дисковых версий БейсикА. Важно, чтобы Бэйсик-ПЗУ оставался неизменным, поскольку это может

отразиться на особенностях выполнения широко распространенных программ, написанных на БэйсикЕ, и никем не поддерживаемых. Возможна ситуация, когда одни и те же Бейсик-программы будут по разному работать на различных машинах только из-за различия в версиях ПЗУ.

Придерживаясь мнения на корректировку исправления Бэйсик-программ дискетной версии, фирма IBM обеспечила пользователю возможность отслеживать все изменения, вносимые в язык. Если каккая-нибуть программа написана в расчете на оригинальный Бэйсик, то следует лишь установить дискету с этой версией БэйсикА. Для новых версий БэйсикА программа может имитировать модификацию содержимого ПЗУ с помощью методики, описанной в главе 3 при осуждении прерываний.

Все изменения, внесенные в ПЗУ, касались BIOS; ниже мы перечислим их и Вы сможете оценить степень их важности. Если вы располагаете первой редакцией "Руководства по техническому обслуживанию IBM/PC", то можете убедиться в том, что текст BIOS-ПЗУ, приведенный в приложении А, соответствует первой версии ПЗУ, о чем свидетельствует также маркер версии на последней странице: 04/24/81. Вслед за текстом BIOS-ПЗУ приведены некоторые замечания. Первое, третье и пятое замечания касаются некоторых ошибок, найденных в BIOS и исправленных во второй его версии.

Первые три изменения, внесенные в BIOS-ПЗУ, касаются программ самотестирования, автоматически запускающихся после включения питания на IBM/PC. Эти программы выполняют четырнадцать различных тестовых проверок правильности работы оборудования, прежде чем пользователь натворит "дел" на неисправном компьютере.

Тест номер три (входящий в состав этих программ) осуществляет проверку правильности работы функции таймера котроллера памяти. Два исправления из трех, внесенных в тестовые программы, касаются этой процедуры. (Здесь представляется уместным показать, что программы, подобные BIOS, работающие непосредственно с аппаратурой, отличаются от обычных программ. Корректность функционирования программ тестирования устройств, а также программы BIOS часто зависит оттонкостей функцмонирования контролируемых устройств. Временные характеристики управляющих сигналов, а также их взаимная согласованность могут оказывать решающее воздействие на факт успешного завершения этих программ. В целом ряде случаев отладка таких программ производится опытным путем - одного логического анализа бывает недостаточно. Вот почему, говоря об изменениях, внесенных в тестовые программы, я не смог сказать ничего определенного о смысле этих изменений).

Одно из изменений, внесенных в программы тестирования,

затрагивает сразу три программы. Цель изменения: обеспечение безусловного обновления операций таймера до момента обработки управляющего прерывания видеомонитора. В первой версии этого не делалось.

Не следует огорчаться, если Вам непонятно значение введенных изменений - они практически никак не сказываются на работоспособности правильно функционирующей системы. Остальные изменения (описываются ниже) представляют для Вас большой интерес.

Одно из внесенных изменений представляет интерес прежде всего для тех, кто использует коммуникационный адаптер. На программы ПЗУ, обслуживающие коммуникационный адаптер, среди прочих, возлагаются также функции оповещения в случае отсутствия на линии связи сигнала отклика. Если по истечении определенного промежутка времени такой сигнал не появился, то коммуникационные программы ПЗУ вырабатывают признак "тайм-аут". /Тайм-аут - интервал времени, когда одна сторона принимает определенные меры, если в течение заданного времени она не получит ответа от другой стороны (Прим.перев.)/. К сожалению, в результате обычной опечатки программы ПЗУ не вырабатывают признак "тайм-аут". Ошибка возникла из-за различий в представлении десятичных и шестнадцатеричных чисел при записи шестнадцатеричного числа 80H символ "H" был потерян и число воспринялось как десятичное (т.е. шестнадцатеричное 50). В программе это число (константа) используется для установки в единичное состояние конкретных битов; таким образом, вместо установки в единичное состояние одного бита-признака состояние "тайм-аут" были установлены два других бита-признака "готовности данных" (условие обратное "тайм-аут") и "ошибка четности" (свидетельствует о возможной ошибке в данных).

Следствием этой ошибки является возможность неправильной работы коммуникационной программы. Если она запрограммирована без учета ошибки BIOS-ПЗУ, то отсутствие ответа в телефонной линии будет восприниматься как ошибка четности данных (таким образом нарушается объективность диагностики).

Следующая ошибка, обнаруженная во второй версии BIOS-ПЗУ, относилась к программам управления принтером. Проблема, как и прежде, заключалачь в тестированипи состояния "тайм-аут". При выполнении какой-либо работы для компьютера принтер может находиться в состоянии "устройство занято"; возможен также такой режим работы, когда принтер сигнал ответа не вырабатывает. Управляющая программа BIOS-ПЗУ тестирует эти режимы, ожидая некоторое время поступления

сигнала ответа. К сожалению, это время было выбрано без учета наиболее длительной операции, выполняемой принтером - протяжки бумаги на интервал, равный одной одной странице. В первой редакции BIOS-ПЗУ это время было явно меньше требуемого, поэтому в новой версии до момента появления признака "тайм-аут" проходит вдвое большее время.

Вслед за принтером изменения затронули и программы поддержки накопителя на магнитной ленте кассетного типа. Здесь была изменена последовательность выполнения нескольких команд, что привело к небольшому изменению временных характеристик процесса считывания данных. Поскольку в составе IBM/PC накопитель кассетного типа используется весьма редко (т.е. последствия ошибки незначительны), то это изменение относится к разряду несущественных.

Последнее изменение представляет интерес для тех пользователей IBM/PC, которые используют цветной графический адаптер. Не всем известно, что при работе с цветным графическим адаптером в графическом режиме (в отличие от работы в текстовом режиме) символы выводятся на экран в режиме рисования, подобно тому как выводится изображение любого другого объекта графической природы. (Эти вопросы подробно рассматриваются в главе 9, посвященной отображению графических объектов). В ПЗУ предусмотрена специальная таблица, содержащая графические образы символов ASCII с тем, чтобы их можно было вывести как элементы алфавита.

При работе цветного графического монитора в текстовом режиме отображаемые символы генерируются цветным графическим
адаптером схемно; в графическом же режиме символы рисуются на экране управляющей программой ПЗУ, использующей табличные образы. Один из этих образов в первой версии BIOS был закодирован с ошибкой, а именно: символ CHR$(4), обозначающий сокровища на колодах для игральных карт, имел одну лишнюю точку снизу. Во второй версии BIOS-ПЗУ эта ошибка устранена.

Ни одна из перечисленных выше ошибок не относится к разряду катастрофических. Наиболее значимая из них влечет за собой неправильное формирование признака "тайм-аут" при работе с асинхронной линией связи , то есть ошибочное описание ситуации, а не ошибочные действия. Редкий пользователь IBM/PC сумеет ее распознать, даже, если знает в чем дело.

Знать о всех изменениях имеет смысл постольку, поскольку нельзя быть уверенным в том, что изменения, затрагивающие особенности работы машинных программ (даже в незначительной степени) всегда должным образом документируются и открыто публикуются. Здесь важен прецедент, дающий основание

беспокоиться, что более важные и существенные изменения могут быть осуществлены без ведома пользователей.

6.6. Механизм выборки информации из ПЗУ

В данном параграфе мы обобщим все аспекты, связанные с активизацией служебных программ BIOS-ПЗУ. Любой вид обслуживания обеспечивается через механизм прерываний; по причинам, котрорые мы подробно рассматривали в главе 3, прерывания позволяют запускать (выполнять) программы, без предварительных сведений о их местоположении(при необходимости можно осуществлять перемещение этих программ).

Ниже приведена таблица прерываний, используемых для активизации программ BIOS-ПЗУ, а также программ, имеющих к ним непосредственное отношение.

Номер	Шестнадцатиричный код	Назначение
5	5	Операция печати содержимого экрана
16	10	Операции ввода/вывода на экран дисплея
17	11	Контроль оборудования; используется для определения подключенного обо-оборудования
18	12	Проверка памяти;используется для определения объема памяти
19	13	Операции ввода/вывода с накопителей на дискете; различные операции с накопителем на дискете; различные опе-

		рации с дискетой
20	14	Ввод/вывод для RS-232; операции с асинхронным коммуникационным адаптером
21	15	Ввод/вывод для накопителя на магнитной ленте кассетного типа
22	16	Ввод/вывод с клавиатуры
23	17	Ввод/вывод для принтера
24	18	Загрузка кассетной версии Бэйсика
25	19	Перегрузка системы
26	1A	Функции даты и времени

Полный список программ обслуживания значитель шире. При необходимости каждой отдельной функции обслуживания мог бы быть присвоен свой собственный номер прерывания или наоборот - все функции могли бы быть обозначены одним кодом прерывания, образуя своего рода управляющиее прерывание.

Более продуктивным, однако, представляется подход, когда с каждым прерыванием связывают один функциональный разрез, обеспечивая таким образом их логическую группировку. Такой путь ведет к управляемому числу отдельных прерываний с обоснованным разделением функций обслуживания.

Несмотря на то, что потребности в каждой функции различны, существуют некоторые общие положения относительно особенностей перемещения данных между точкой вызова функции и собственно функций. Если Вы используете интерфейсные программы, поставляемые на дискете в качестве дополнения к данной книге, у Вас не будет возможности ознакомиться с деталями реализации каждой функции. Знать как они работают между тем весьма полезно.

За исключением отдельных случаев в рамках одного прерывания реализуются несколько функций. Для передачи номера, обозначающего субфункцию используется регистр AH. Для передачи параметров в обоих направлениях используются

регистры, причем система стремится к их экономному использованию. Программы обслуживания обеспечивают сохранение
всех регистров, не используемых для передачи параметров; вызывающая программа таким образом не должна заботиться о сохранении величин, находящихся в различных регистрах. Если требуется выделить область памяти для данных, например, для буфера данных считываемых с дискеты или кассеты, то для представления сегментированных адресов используются регистры ES и BX.

Регистры AH, AX мспользуются для передачи программам обслуживания кода возврата или результирующего сигнала. В некоторых случаях для индикации ошибки используется флаг переноса CF, однако как правило флаги не используются. Регистры общего назначения AX ... DX следует использовать экономно по мере необходимости, придерживаясь алфавитного порядка.

В последующих пяти главах мы детально рассмотрим все перечисленные выше типы прерываний и особенности работы с регистрами.

6.7. Описание специальных прерываний

Как уже упоминалось в главе 3, три строки в таблице векторов прерываний используются не по прямому назначению для хранения адресов программ обслуживания прерываний , а для хранения адресов базовых таблиц системы (поскольку эти таблицы размещаются в ПЗУ). Ниже мы поясним назначение этих таблиц.

Первая таблица (с кодом фиктивного прерывания 29) содержит адрес таблицы инициализации видеодисплея, используемой для контроля управляющих кодов для экранов дисплеев. Таблица размещается в ПЗУ по адресу F000: F0A4 и включает три подтаблицы с видеопараметрами. Каждая подтаблица имеет четыре строки по одной строке на каждый из четырех режимов отображения: монохромный, 40-столбцовый цветной, текстовой, 80-столбцовый цветной, текстовой и оба графических режима. Одна из подтаблиц фиксирует размер в байтах буфера дисплея, соответствующего текущему режиму. (Подробно эти вопросы рассматриваются в главе 8 и 9).

Еще одна подтаблица фиксирует ширину граф, учитываемую при автоматическом переходе от одной строки в другую. Практическая польза от модификации этой таблицы близка к

нулю, поэтому в дальнейшем она рассматривается не будет.

Адрес второй таблицы (соответствует прерыванию 30) указывает на таблицу параметров дискет. Эта таблица содержит данные, предназначенные для контроля временных параметров различных операций, выполняемых с дискетой. Таблица расположена в ПЗУ по аресу F000 : EFC7. Сразу же после выпуска компьютера IBM/PC стало ясно, что некоторые временные характеристики, внесенные в эту таблицу весьма приблизительны и являются причиной слишком медленной работы дискового накопителя. Среди многочисленных изменений, внесенных в DOS версии 1.10 были данные, снижающие время обращения к дисковому накопителю. Здесь использовался классический метод - в процессе самонастройки дисковая операционная система создавала новую версию таблицы в обычной памяти и заменяла ее адрес в векторе,инициализируцемой по прерыванию 30. Поскольку при выполнении любой операции с диском производится обращение к этой таблице, то после замены адреса вектора, активной становилась новая таблица. В главе 7 (посвященной описанию дискового накопителя) мы рассмотрим детали, связанные с этой таблицей и укажем внесенные изменения.

Адрес третьей таблицы, получаемой по фиктивному прерыванию 31, указывает на область генератора графических символов, используемых при формировании на дисплее символов CH$(128), ..., CHR$(255) в любом из двух графических режимов. Стандартной таблицы этого типа попросту не существует. Если две предыдущие таблицы всегда находятся в ПЗУ, то данная таблица в ПЗУ отсутствует. А коль скоро это так, то вектор прерывания будет содержать нулевое значение (является признаком отсутствия таблицы) до тех пор, пока программа не построит такую таблицу и не занесет соответствующий адрес в вектор прерывания. При изучении графического режима работы (глава 9) мы увидим, что существует много интересных вещей, связанных с таблицей генерации символов.

Все эти общие положения, касающиеся ПЗУ, образуют ту основу, опираясь на которую мы перейдем к рассмотрению широких возможностей отдельных узлов IBM/PC. Этому посвящены пять следующих глав. Начнем с дисковых накопителей, рассматривая их до того как программа построит таблицу и поместит адрес этой таблицы в соответствующий вектор прерываний.

Приложение 6.1. Текст программы проверки метки версии ПЗУ (Паскаль).

```
module Listing_6_1;


type

  string8       = string(8);
  bios_ptr_type = ads of string8;


var

bios_date    : bios_ptr_type;


procedure check_rom_bios_date;

  begin

    bios_date.s :=#F000;
    bios_date.r :=#FFF5;

    if bios_date^='04/24/81' then
      begin

      end
    else
      begin

      end;

  end;

end.
```

ГЛАВА 7. ДОСТУП К ДИСКЕТАМ

[Оглавление]

С этой главы начинается подробное по-элементное рассмотрение возможностей IBM/PC - начиная с дисководов для гибких магнитных дисков. Изложение материала этой и четырех последующих глав будет следовать организации служебных процедур системы BIOS в ПЗУ. Однако, наше изложение не ограничивается только описанием возможностей, которые

предоставляются системой BIOS. Для каждой из возможностей будет рассматриваться вся техническая информация, полезная или интересная любому, кто хочет понять особенности реализации этой возможности и особенности ее использования.

Если обратиться к трем уже упоминавшимся областям интересов, то следует отметить, что вся информация полностью относится только к IBM/PC. Что касается совместимых с IBM/PC компьютеров, можно ожидать, что большая часть информации применима и к ним тоже, но степень применимости будет зависеть от того, насколько точно конкретный компьютер копирует IBM/PC; для всех компьютеров, которые рекламируются как совместимые по системе BIOS , данная информация полностью применима. Для семейства компьютеров, работающих под управлением операционной системы MS-DOS, приведенная в этих главах информация будет типичной с точки зрения возможностей, которые должен предоставлять любой компьютер, но нельзя рассчитывать, что все детали полностью совпадут.

7.1. ТРИ УРОВНЯ ДОСТУПА К ДИСКЕТЕ

Имеются три способа, позволяющие осуществлять доступ к данным, хранящимся на дискете, - из программы на языке программирования высокого уровня (такого, как Паскаль или БЕЙСИК), посредством вызова функции ДОС и посредством обращения к служебной процедуре системы BIOS в ПЗУ. Все эти способы образуют трехуровневую иерархию, в которой служебные процедуры системы BIOS образуют нижний, наиболее примитивный уровень. Уровень функций ДОС строится на основе служебных процедур системы BIOS. Уровень языков программирования является верхним уровнем и строится на основе функций ДОС и служебных процедур системы BIOS.

Может существовать еще один, более высокий уровень, обеспечиваемый системой управления базами данных того или иного типа.

Операции ввода/вывода с дискетой на уровне языков программирования обычно удовлетворяют все потребности по манипулированию данными, хранящимися на дискете. В тех случаях, когда этого оказывается недостаточно, не хватает как раз не "атомарных" функций низкого уровня, а процедур более высокого уровня, прредоставляемых только системами управления базами данных. Однако, иногда требуется доступ к дискете на более низком уровне, например, для прямого чтения или записи секторов дискеты. Для выполнения таких операций необходимо обращаться к двум другим уровням ввода/вывода с дискетой - к уровням ДОС И BIOS.

В главе 4 мы уже рассмотрели доступ ко всем функциям

ДОС, включая и функции ввода/вывода с дискетой. Одно из главных достоинств функций ввода/вывода на уровне ДОС заключается, как Вы могли уже заметить изучая главу 4, в ширине диапазона их возможностей. Функции ДОС позволяют выполнять как операции довольно высокого уровня (например, поиск в справочнике файлов по родовым именам), так и операции среднего уровня (чтение и запись логических записей) и даже операции низкого уровня (чтение и запись секторов по абсолютному номеру).

Широкие возможности уровня функций ДОС делают их особенно привлекательными для использования. Языки программирования, такие как Паскаль, обычно организуют все свои средства ввода/вывода с помощью функций ДОС. Если Вы хотите освободиться от ограничений, накладываемых организацией системы ввода/вывода выбранного Вами языка программирования, я советую обратиться к функциям ввода/вывода ДОС.

Если главная причина обращения к специальным операциям ввода/вывода связана с необходимостью чтения или записи секторов дискеты, то лучше воспользоваться функциями ДОС, чем процедурами BIOS. Это связано только с тем, что дополнительный сервис, предоставляемый функциями ДОС, избавит
Вас от необходимости заботиться о таких деталях как повторение попытки выполнения операции в случае ошибки или выдержки времени, необходимого для разгона двигателя дисковода. Тем не менее, Вам может потребоваться прямой доступ к процедурам уровня BIOS, например, если нужно обойти процедуры обнаружения ошибок ДОС или когда нужно выполнить операцию, которая не выполняется средствами ДОС. По этой причине ниже приводится описание служебных процедур системы BIOS в ПЗУ, предназначеннных для работы с дискетами.

7.2. Служебные процедуры обслуживания дискет системы BIOS в ПЗУ.

Для обслуживания дисководов в системе BIOS имеется шесть процедур. Доступ ко всем осуществляется с помощью прерывания номер 19 (шестнадцатиричное значение 13). В этом разделе мы рассмотрим каждую из них и увидим некоторые примеры их использования. Программы 7.101 и 7.102, включенные в дисковый пакет прилагающийся к этой книге, обеспечивают необходимые ассемблерные подпрограммы для доступа к этим процедурам BIOS, а также определения и вспомогательные программы на Паскале, облегчающие использование ассемблерных подпрограмм.

Первая служебная процедура, которая имеет код 0, сбрасывает дисковую подсистему в исходное состояние. Эта процедура может использоваться для восстановления после различных ошибок и сбоев. Эта операция аналогична операции сброса в ДОС (вызов функции номер 13), но она выполняется на более ниэком уровне системы BIOS. Операция сброса диска посылает команду контроллеру дисков, приводящую его в исходное состояние. Заметим, что здесь не выполняются такие действия как установление дисковода, выбираемого ДОС по умолчанию, - для этого должна выполняться операция сброса на уровне ДОС.

Наиболее очевидное использование операции сброса - в процедуре восстановления после ошибки. Часто наилучшим способом действий программы в случае ошибки операций с дискетой является быстрое прекращение всех действий, с предоставлением возможности разбираться в происшедшем пользователю компьютера. В конце концов, большинство дисковых ошибок связаны с причинами, которые невозможно устранить программно, например, с механическими повреждениями дискеты или неправильной работой механики дисковода. Однако, Ваша программа должна использовать все имеющиеся возможности для восстановления после ошибки и процедура сброса может стать важной составной частью такой программы восстановления.

Вторая служебная процедура, с кодом 1, позволяет прочитать код состояния дисковой подсистемы. Состояние изменяется любой дисковой операцией, так что эта процедура отражает последнюю выполненную операцию с дискетой. Эту процедуру можно использовать для слежения за теми операциями ввода/вывода, которыми Вы не можете управлять непосредственно. Например, если выполняется обычная операция ввода/вывода на уровне языка высокого уровня и получено сообщение об ошибке, вызов этой процедуры поможет лучше понять, что произошло, в чем состоит ошибка, так чтобы программа могла выполнить необходимые корректирующие действия. Ниже приведены коды состояний, которые могут возвращаться, сами по себе или в некоторой комбинации, этой процедурой.

КОД	ШЕСТНАДЦАТИРИЧНОЕ ЗНАЧЕНИЕ	СМЫСЛ
1	01	Была выдана неправильная команда
2	02	Не найден адресный маркер (используемый для обнаружения сектора)

3	03	Была запрошена операция записи для дискеты, защищенной от записи
4	04	Запрашиваемый сектор не найден
8	08	Неправильный адрес при операции прямого доступа к памяти (ПДП)
9	09	При ПДП перейдена граница 64К памяти
16	10	Ошибка чтения данных, обнаруженная при проверке циклического избыточного кода
32	20	Ошибка контроллера гибких дисков
64	40	Ошибка поиска указанной дорожки
128	80	Тайм-аут: ответ от дисковода не получен в положенное время

Третья и четвертая процедуры, с кодами 2 и 3, считывают и записывают один или несколько секторов (объединяемых в блок) на одной дорожке. Я советую использовать эквивалентные функции ДОС, что избавит Вас от необходимости проверять возникновение ошибок или повторять выполнение операций. Заметим что эти две операции, а также соответствующие функции ДОС позволяют считывать или записывать одновременно несколько секторов. Если Вам необходимо выполнить очень быстрый ввод или вывод данных, эти операции позволяют переслать вплоть до целой дорожки данных за время одного оборота дискеты. Чтение/запись секторов по одному требуют затраты времени одного оборота на каждый сектор.

Впрочем чтение/запись нескольких секторов имеет и свои недостатки. Один из таких недостатков - необходимость организации большого буфера, размером до 4К байт при чтении целой дорожки. Если можно работать с данными в том же месте памяти, куда они считываются или откуда они записываются, такой недостаток не будет существенным, но если требуется выделить отдельную буферную область для работы с дискетой, тогда чтение или запись нескольких секторов увеличат требования к памяти, занимаемой Вашей программой. Имеются и другие недостатки, например, если происходит ошибка, то ее обнаружение и восстановление после нее могут оказаться значительно сложнее, поскольку трудно установить на каком из секторов произошла ошибка. Эта сложность несколько меньше при использовании служебных процедур BIOS, поскольку они ведут подсчет числа переданных секторов, который может быть проанализирован программой; функции ДОС не обеспечивают такой возможности. Проблема ошибки в процессе выполнения многосекторной операции не слишком существенна при чтении, но

при записи она может весьма осложнить задачу восстановления, если программа не может определить сколько секторов уже было записано.

Пятая служебная процедура, с кодом 4, используется для верификации данных после операции чтения или записи. Она повторно считывает сектора, используя для проверки циклические избыточные коды. Эта процедура используется по ключу "/V" в команде ДОС COPY. Верификацией не следует злоупотреблять и на то есть все основания. Во-первых, операции с дискетами достаточно надежны. Во-вторых, практически все ошибки при пересылке данных обнаруживаются и сообщения о них передаются программам; очень редко операция чтения или записи завершается успешно, а данные при этом как-то искажаются. Однако, когда гарантия безошибочности действительно важна, необходимо выполнять верификацию после пересылки данных. Эта операция, помимо всего прочего, значительно увеличивает время выполнения операции. Для верификации чтения или записи сектора эту процедуру следует вызывать сразу же после завершения операции пересылки.

Шестая, и последняя, операция, с кодом 6, особенно интересна. Эта процедура форматирует дорожку дискеты, записывая адресные маркеры секторов и заполняя сегмент данных каждого сектора стандартным шестнадцатиричным значением F6 или CHR$(246). Должны указываться отдельно для каждого сектора такие спецификации как дорожка, сторона и номер сектора, а также код, задающий длину сектора. Поскольку размер каждого сектора может быть задан независимо от остальных, можно сформатировать дорожку с одним или несколькими секторами нестандартного размера - что является обычным средством для защиты от копирования. Обычными средствами ДОС нельзя прочитать сектор, размер которого отличается от стандартного 512-байтного, поэтому многие схемы защиты от копирования пользуются именно такими средствами. Более подробно мы рассмотрим это в следующем разделе, посвященном параметрам, управляющим операциями с дискетой.

Форматировать нужно целую дорожку сразу - поскольку промежутки между секторами и адресные маркеры взаимозависимы.

Однако, если необходимо переформатировать всего один сектор, способ для этого все же есть. Например, если нужно изменить формат последнего сектора на дорожке, сохранив данные в первых семи секторах, поступите так: напишите программу для чтения первых семи секторов, затем переформатируйте всю дорожку и перезапишите данные. Такая процедура позволит защитить от копирования уже существующие данные, хотя внешне вроде бы ничего не изменяется.

Для форматирования всех секторов на дорожке должны быть заданы следующие параметры, каждый в виде однобайтного числа:

1 - номер дорожки (от 0 до 39)
2 - сторона (0 или 1; для односторонних дискет всегда 0)
3 - номер сектора (начиная с 1; сектор 0 означает управляющую информацию)
4 - код длины (0=128 байт, 1=256 байт, 2=512 байт [стандартное], 3=1024)

Поскольку каждый сектор имеет собственный код длины, включить сектор нестандартной длины в дорожку не представляет особого труда (либо создать целую дорожку из секторов нестандартной длины для ДОС).

Если для форматирования дорожки используется процедура уровня BIOS, следите за правильным указанием всех параметров.

7.3. Параметры дискеты и защита от копирования

Работа дисковода для гибких дисков частично управляется таблицей, известной как база дисков или таблица параметров дискеты (терминология фирмы "ИБМ" пока не устоялась)

База дисков - это таблица, состоящая из одиннадцати байт, которая содержит ряд параметров, необходимых для работы дисководов, включая и определенную информацию о формате дискет. В листинге 7.1 можно найти определение этой таблицы, комментарии к которому описывают назначение каждого байта.

Как упоминалось в главе 6, один вектор прерывания,- номер 29, хранящийся в ячейке с адресом 120,- используется для указания местонахождения этой таблицы. Исходная версия этой таблицы, использовавшаяся версией ДОС 1.00, находилась в ПЗУ вместе с системой BIOS, и поэтому ее можно видеть в листинге BIOS, приведенном в "Техническом руководстве".

Начиная с версии ДОС 1.10 вместо этой таблицы стала использоваться таблица, размещаемая в оперативной памяти. Такое изменение, кстати, дает нам одно преимущество. Это позволяет экспериментировать со значениями в таблице, не прибегая к различным ухищрениям для подмены таблицы в ПЗУ. Любая программа может установить собственную базу диска, но вот создать ее таким образом, чтобы эта таблица продолжала использоваться после окончания программы, довольно сложно. Для достижения это цели необходимо зарезервировать небольшую область памяти так, чтобы ДОС не использовала ее. Однако, с тех пор как ДОС стала размещать таблицу базы диска в

оперативной памяти, во всех этих приемах нет необходимости.

По сравнению с версией ДОС 1.00 в версии 1.10 сделано всего два изменения в таблице базы диска, но они позволили значительно увеличить скорость использования дискет. Во-первых, они на 25% уменьшили время, выделенное дисководу для перемещения головки с дорожки на дорожку. Это время получило название "время шага" (SRT). В версии таблицы, размещаемой в ПЗУ, которая использовалась в ДОС версии 1.00, время шага равнялось 8 миллисекундам. Новое значение, использующееся начиная с ДОС 1.10, равняется 6 миллисекундам.

Этот параметр в таблице занимает первый полубайт. Его исходное значение было 12 (шестнадцатиричное значение C), а новое значение 13 (шестнадцатиричное значение D). Я сначала предположил, что такое изменение значения означает увеличение времени шага, однако Роберт Бэттен сообщил мне, что используется инверсное значение этого параметра, так что большее значение означает меньшее время шага. Время шага управляется приращениями по 2 миллисекунды, так что изменение времени шага с 12 на 13 в таблице соответствует уменьшению действительного времени с 8 до 6 миллисекунд.

Второе изменение резко увеличило скорость доступа к дискете. "Время установления головки", то есть, пауза, необходимая для стабилизации головок чтения/записи, была уменьшена с 25 миллисекунд до нуля. Первоначально фирма "ИБМ" чересчур предубежденно относилась к быстродействию дисководов, так что за счет ликвидации времени установления удалось добиться сразу большого увеличения быстродействия. Большую часть параметров этой таблицы изменять нельзя, однако, некоторые из них предоставляют широкие возможности для экспериментаторов. В девятом байте хранится значение, которое используется операцией форматирования для инициализации секторов дискеты. Стандартное значение F6, но Вы можете его изменить. Некоторые программы проверяют наличие кода F6, чтобы найти неиспользовавшиеся сектора, так что это значение вряд ли разумно изменять, если для этого нет веских оснований. Такое изменение целесообразно применять для защиты от копирования.

Еще один интересный параметр - это байт размера сектора. Он определяет размер сектора, который должен считываться или записываться. Это позволяет работать с секторами, размер которых отличается от стандартного 512-байтного формата ДОС. В других операционных системах могут использоваться другие размеры секторов за счет изменения этого байта. Изменение этого значения является наиболее простым способом организации защиты записи. Для формирования сектора с размером, отличным от 512 байт, необходимо использовать служебную процедуру

форматирования, описанную в предыдущем разделе. В процессе форматирования размер каждого сектора определяется отдельно. Однако, чтобы в дальнейшем Ваши программы могли читать или записывать нестандартные сектора, необходимо изменять указатель размера сектора в таблице базы диска.

Этот параметр может принимать одно из следующих значений:

Значение	Размер сектора в байтах
0	128
1	256
2	512 (стандартный для ДОС)
3	1024

Листинг программы 7.1 демонстрирует (на Паскале), как осуществляется доступ к таблице базы диска и как изменяются ее параметры. То же самое можно сделать на БЕЙСИКе, хотя здесь требуется побайтное извлечение адресных значений. Ниже приведена программа на БЕЙСИКе, которая находит и распечатывает таблицу базы диска.

```
100  REM  Программа на БЕЙСИКе для поиска и распечатки
       таблицы базы диска
110  REM
120  REM  Сначала необходимо установить указатель вектора
       прерываний
130  REM
140  DEF  SEG=0    'начало памяти
150  OFFSET1=30*4  'смещение до вектора прерывания
160  REM
170  REM  Далее нужно определить смещение адреса вектора
180  REM
190  OFFSET2=PEEK(OFFSET1)+256*PEEK(OFFSET1+1)
200  REM
210  REM  Теперь нужно определить сегментную часть
       адреса вектора
220  REM
230  DEFSEG=PEEK(OFFSET1+2)+256*PEEK(OFFSET1+3)
240  REM
250  REM  Теперь можно начинать просмотр таблицы
260  REM
270  PRINT "Ниже следуют данные талицы базы диска в
         шестнадцатиричном виде"
280  FOR I=0 TO 10
290   PRINT " ";
```

```
300   IF PEEK(OFFSET2+1)=16 THEN PRINT"0"; 'добавмть
                ноль для четности
310   PRINT HEX$(PEEK(OFFSET2+1));
320  NEXT I
330  PRINT:PRINT
```

Если Вам нужны средства для защиты от копирования, мы предоставляем их Вам в двух формах.

Программы 7.103 и 7.104, имеющиеся на дискете, прилагаемой к этой книге, предоставляют процедуры на ассемблере и на Паскале, которые позволят организовать защиту записи и приспособить ее к Вашей схеме защиты. Кроме того, в пакет входит готовая программа защиты, которую можно вызывать как на БЕЙСИКе, так и на Паскале.

```
{  Листинг  7.1  -- процедуры на Паскале для доступа к пара-}
{  метрам "базы диска"                                    }

{ Этот фрагмент программы демонстрирует возможность доступа }
{ к "базе диска" или таблице управления дискетами. Этот     }
{ листинг разрабатывался для обеспечения наиболее легкой    }
{ настройки на конкретные нужды пользователей перед его ис- }
{ пользованием.                                           }

module Листинг 7.1;

type

disk_base_type = array [1..11] of byte;

{ Ниже кратко описаны все 11 байт таблицы : }

{ 1 : время шага; время вывода головки }
{ 2 : время ввода головки; режим ПДП }
{ 3 : время ожидания отключения двигателя }
{ 4 : длина сектора в байтах: 0=128,1=256,2=512,3=1024 }
{ 5 : конец дорожки (номер последнего сектора) }
{ 6 : длина межсекторного промежутка }
{ 7 : длина сегмента данных (когда не задана длина сектора) }
{ 8 : длина промежутка для форматирования  }
{ 9 : байт заполнитель для форматирования  }
{ 10: время установления головки           }
{ 11: время запуска двигателя             }
```

```
var

 disk_base         : disk_base_type;
 disk_base_pointer  : ads of disk_base_type; {сегментированный
                                  адрес}
 vector_pointrer    : ads of adsmem; {сегментированный адрес}

procedure disk_base_access;
  begin
{сначала указатель устанавливается на вектор прерывания базы}
{ диска}
    vector_pointer.s := 0; {раздел сегмента 0, начало памяти}
    vector_pointer.r := 30 * 4; {смещение для вектора 30}

{далее считывается вектор, чтобы установить местоположение}
{ таблицы}

    disk_base_pointer := vector_pointer ^;
{указатель базы диска ("disk_base_pointer") теперь содержит }
{сегментированный адрес, на который указывал указатель век- }
{тора ("vector_pointer") -- это означает, что указатель базы}
{диска теперь укаазывает на адрес фактического местонахожде-}
{ния таблицы                                  }

{Теперь будет выполняться проверка того. находится ли таблица}
{в оперативной памяти. Это можно установить попытавшись из-  }
{менить значение в таблице и затем проверив, изменилось ли   }
{оно на самом деле. Вместо этого мы просто проверим, в каких }
{адресах памяти располагается таблица.                }

    if disk_base_pointer.s >= #F000 then
      begin
{адрес слишком велик - таблица в ПЗУ и изменить ее не удастся}
      end
    else
      begin
{все в порядке - таблица наверняка в ОЗУ}
      end;

{Теперь скопируем таблицу в свой буфер }
  disk_base := disk_base_pointer ^;
{База диска теперь содержит копию данных таблицы, на которую}
{указывал сегментированный адрес в указателе базы диска }

{Далее, изменим размер на 1024 байта и байт-заполнитель для}
{форматирования на AA. Эти изменения мы выполняем в своей  }
```

```
{копии, которую затем переносим в реальную таблицу. Впрочем,}
{то же самое изменение можно было бы выполнить и непосред-  }
{ственно в самой таблице.}

  disk_base [4] := 3; {установить размер сектора 1024 }
  disk_base [9] := #AA {шестнадцатиричное значение AA }
                {или десятичное 170 выбрано произвольно}

  disk_base_pointer ^ := disk_base; {возвратить таблицу на }
                              {место}
  end;
end.  {конец модуля listing_7_1 }
```

ГЛАВА 8. ВИДЕОДОСТУП, ЧАСТЬ 1 - ТЕКСТОВЫЙ РЕЖИМ

[Оглавление]

Практически любая программа, выполняемая на персональном компьютере фирмы IBM выводит информацию на экран видеодисплея. Существует два принципиально различных вида информации,представляемой на экране дисплея IBM/PC: текст (т.е. литеры алфавита и другие символы) и графика. В настоящей и в следующей главах мы рассмотрим все наиболее существенные аспекты, связанные с доступом к содержимому экрана, расмотрение начнем с символьных текстов.

В отличие от остальных глав данной книги, эти главы практически не содержат сведений, общих для всего семейства компьютеров, использующих операционную систему MS-DOS. В то же время рассматриваемые материалы помогут нам установить степень совместимости всего класса персональных компьютеров с персональным компьютером IBM/PC, поскольку большинство распространенных программ для IBM/PC зависят от специфики представления информации на экране.

8.1. Типы экранов видеодисплеев

Существует множество различных типов видеодисплеев, подключаемых к компьютеру IBM/PC, обладающих различными характеристиками и особенностями программирования. Для того, чтобы разобраться в этом вопросе, мы опишем наиболее существенные различия и покажем, как они соотносятся с теми сведениями, которые мы приводим в настоящей и следующей главах.

Прежде всего, следует отметить, что к персональному компьютеру в качестве дополнительных плат могут быть

подключены два дисплейных адаптера - одноцветный (монохромный) и цветной графический. Дисплейный адаптер связывает компьютер с монитором дисплея с помощью микросхемы, называемой контроллером электронно-лучевой трубки (экрана). Адаптер имеет также ряд программируемых портов ввода-вывода, генератор символов в ПЗУ и оперативную память, которая хранит информацию, выводимой на экран информации.

Изображение на экране, как уже указывалось выше, генерируется в двух основных режимах - текстовом и графическом. В текстовом режиме на экран выводятся только символы, хотя многие из этих символов можно использовать для построения простых линейных конструкций. Графический режим используется главным образом для представления сложных изображений (схем, графиков, диаграмм, рисунков и т.п.), хотя и допускает возможность представления символов различного размера и формы (формата).

Одноцветный адаптер предназначен для использования только с монохромным экраном дисплея фирмы IBM. Монохромный
дисплей не может воспроизводить цвет и графику. Он предназначен только для работы в текстовом режиме, т.е. может представлять только текстовые символы и простейшие конструкции (диаграммы), синтезируемые из символов, записанных в ПЗУ в коде ASCII. Многие пользователи предпочитают цветному графическому дисплею монохромный (одноцветный), поскольку с него легче производить считывание информации. Но, выбирая монохромнвй дисплей, они жертвуют графическими возможностями и цветом - двумя очень ценными качествами дисплея.

Цветной графический адаптер может работать с различными типами дисплеев. Он имеет выходные гнезда для двух видов подключений и поэтому допускает подключение разнообразных видеотерминалов. На одном из выходов формируется единый составной сигнал цветного изображения. На другом - отдельные сигналы для трех основных цветов - красного, зеленого и синего; эти сигналы получили обобщенное название - RGB-сигнала. Этот адаптер может работать как в текстовом, так и в графическом режимах, формируя на экране изображения и символы нескольких форматов и различных цветов. Он разработан для работы с любыми видами дисплеев от обычного телевизора до цветного монитора с высокой разрешающей способностью.

К RGB-выходу подключаются специальные дисплеи, известные под названием RGB-мониторов. К выходу составного сигнала могут быть подключены цветные дисплеи соответствующего типа. К этому же выходу подключаются и обычные цветные телевизоры, снабженные частотным преобразователем. Все перечисленные

здесь типы дисплеев способны воспроизводить цвет.

К выходу составного сигнала можно также подключать черно-белый компьютерный дисплей. В этом случае на вход дисплея поступает сигнал цветности, который, однако, не может быть в полной мере воспроизведен средствами дисплея. При использовании такого дисплея теряется возможность воспроизведения цвета, а также высокое качество и четкость монохромного дисплея фирмы IBM. Это самый дешевый способ отображения информации на экране.

Для преодоления ограничений, присущих одноцветному адаптеру, некоторые фирмы-изготовители перешли к выпуску разновидностей одноцветного адаптера, таких, например, как популярный дисплейный адаптер Геркулес, который сочетает графические (но не цвет) возможности цветного графического адаптера с высоким качеством воспроизведения текста, характерным для одноцветного адаптера. В результате качество графики оказывается выше качества, даваемого цветным графическим адаптером. Усиленный графический адаптер может создавать подобную графику на одноцветном экране.

Приблизительно две трети всех персональных компьютеров оборудованы стандартными одноцветными адаптерами и поэтому не
обладают графическими и цветовыми возможностями.Не смотря на определенные реальные преимущества использования цвета и графики, большинство персональных компьютеров прекрасно обходятся без них. Планируя использование персонального компьютера следует помнить, что большинство дисплеев - текстовые.

Различные аспекты излагаемой проблемы в настоящей и следующих главах в той или иной степени касаются использования различных типов дисплеев. Для того, чтобы избежать возможных недоразумений, кратко опишем их возможности.

Любые сведения, относящиеся к одноцветному адаптеру, касаются только этого адаптера. Они не распространяются на различные аспекты воспроизведения цвета, включая использование черно-белого монитора с цветным графическим адаптером. Черно-белый монитор можно рассматривать в качестве монохромного; он однако не используется с одноцветным адаптером. Термин "монохромный" (одноцветный) используется только для монохромного экрана фирмы IBM.

Специфика работы цветного графического адаптера не применима к монохромному дисплею. Все цветные дисплеи разрабатываются с учетом этой специфики. Большинство сигналов цветного графического адаптера воспринимается черно-белым монитором; исключение составляют сигналы цветности.

Большинство черно-белых мониторов обрабатывают сигналы цветности неудовлетворительно. Черно-белые мониторы, формирующие осмысленное изображение при получении цветных сигналов большая редкость. Поэтому любая программа, которая может быть использована на компьютере с черно-белым монитором не должна использовать цвет; в противном случае она должна обладать способностью преобразовать любой цвет в черно-белую палитру.

Цветные телевизоры, как правило, имеют более низкую разрешаемую способность по сравнению с обычными цветными мониторами компьютеров. Поэтому цветные телевизоры могут работать только в режиме самой низкой разрешающей способности, предусмотренной в IBM/PC, а именно в режиме 40 позиционных строк. В большинстве случаев цветной телевизор не позволяет использовать всех возможностей IBM/PC; наиболее важные программы не могут выполняться на компьютере, оборудованном цветным телевизором. Как правило цветной телевизор используется программами реализации компьютерных игр, ориентированных на использование интерфейса с накопителем на магнитной кассете.

8.2. Принципы формирования изображения

Прежде чем перейти к дальнейшему рассмотрению текстового режима работы дисплея, мы для уяснения сути дела кратко рассмотрим некоторые принципы представления изображений на экранах видеодисплеев. Существует два основных способа вывода информации из персонального компьютера на экран видеотерминала. В первом случае компьютер рассматривает экран дисплея в качестве обычного устройства ввода/вывода. При таком подходе компьютер выдает контроллеру дисплея различные команды, включая команды вывода текстовой информации. Ниже приводится некоторая упрощенная форма этих команд:

```
ОЧИСТИТЬ ЭКРАН
ВЫСВЕТИТЬ С 15 ПОЗИЦИИ 10 СТРОКИ СЛЕДУЮЩИЙ ТЕКСТ :
"В случае готовности нажмите клавишу "ввод"" ...
ПЕРЕМЕСТИТЕ КУРСОР В 43 ПОЗИЦИю 10-й СТРОКИ
```

Наиболее существенный аспект взаимодействия такого рода состоит в том, что в этом случае дисплей рассматривается просто как и любое другое периферийное устройство. Разумеется, дисплей может выполнять и ряд специфических команд, таких, которые не может выполнять принтер - например, команду "ОЧИСТИТЬ КАДР" - и все же принцип взаимодействия с

компьютером остается неизменным.

Преимущество описанного подхода состоит в том, что использование команд позволяет рассматривать экраны дисплеев в одном ряду с другими устройствами ввода/вывода. Вследствие этого программное обеспечение поддержки видеодисплеев становится проще и более унифицированным; это позволяет универсальным образом использовать множество различных компьютерных терминалов. Недостаток такого подхода в том, что компьютер теряет непосредственную связь с экраном дисплея и оказывается не в состоянии "творить чудеса" на экране. Фирма IBM пошла по пути "сотворения чудес" и поэтому выбрала другой подход подключения дисплея к IBM/PC.

Этот подход известен как подход, связанный с регенерацией изображения на основе образа экрана хранящегося в памяти. В этом случае компьютер и дисплей совместно используют некоторое пространство общей памяти. Электронные схемы дисплея постоянно производят опрос (считывание) памяти и сразу же отображают результаты на экране. При таком подходе компьютерная программа может осуществлять вывод информации на
экран путем простого изменения содержимого памяти. Аналогично, считывание информации с экрана дисплея может осуществляться путем обычного считывания содержимого общей памяти. Таким образом, экран дисплея представляется фактически областью памяти и выборка/занесение информации из(в) память означает считывание или запись информации с(на) экран дисплея.

В персональном компьютере фирмы IBM фактически используется смешанный подход. Для отображения данных на экране используется хранимый в памяти образ экрана (карта) , а для управления различными аспектами отображения используется система команд. В дисплейных системах с жесткой реализацией механизма отображения содержимого памяти даже управление курсором осуществляется путем изменения содержимого общей памяти. В компьютере IBM/PC позиционирование курсора осуществляется с помощью команд, даваемых схемой управления контроллером. Команды используются
в следующих случаях: установки размера мерцающего курсора, очистки и обновления экрана, а также для изменения режима работы дисплея (текстовый-графический, 40 позиционный - 80 позиционный).

Работа в режиме хранимого в памяти образа экрана (карты) связана с двумя потенциальными неудобствами (однако фирме IBM удалось найти способ их преодоления). Первое состоит в том, что этот образ требует для размещения значительных ресурсов

оперативной памяти; для восьмиразрядных компьютеров старого исполнения с малой оперативной памятью это представляло серьезную проблему. В компьютере IBM/PC эта проблема решается в два приема. Во-первых, расширением пространства адресуемой памяти до миллиона байт. Во-вторых, выделением дисплею своей собственной встроенной памяти, с тем чтобы не использовать обычную память, предназначенную для хранения программ и данных.

Существуют и более веские основания для обеспечения дисплейного адаптера своей собственной памятью и следовательно разрешение проблемы конфликтов при доступе к памяти. Дело в том, что дисплей с хранимой копией изображения (образа экрана) в памяти должен постоянно осуществлять операции чтения из памяти и, следовательно, конкурировать за эту память с микропроцессором. В этом случае образуется очередь за обладание схемными ресурсами доступа к памяти.

Фирма IBM решила эту проблему путем выделения дисплею своей собственной памяти и организовав доступ к этой памяти через два порта ввода-вывода. Такой подход обеспечивает возможность быстрого доступа к общему полю памяти и со стороны процессора и со стороны дисплея.

Если внимательно присмотреться к экрану в момент генерации кадра изображения, то можно заметить что большую часть времени экран дисплея остается совершенно чистым, и только в некоторых случаях возникает нечто вроде быстро исчезающей помехи, как если бы курсор попытались быстро переместить через весь экран дисплея. Такой эффект возникает как результат попыток одновременного доступа к памяти со стороны компьютера и со стороны дисплея. Опасного в этом ничего нет, любопытен лишь сам эффект и не более. Взяв за основу дисплей, хранящий копию изображения в памяти, закрепив за дисплеем свою собственную область памяти и обеспечив двусторонний вход в память фирма, IBM решила задачу визуализации информации наилучшим образом. Можно было бы, правда, обеспечить и более высокую разрешающую способность дисплея в текстовом режиме, а также реализовать пиктографический режим, хотя бы в том объеме как это было впервые сделано для рабочей станции "Стар" (STAR) фирмы "Ксерокс" или для компьютера "Лиза" (Lisa) фирмы "Apple".

В базовом варианте распределения памяти для IBM/PC под память дисплея отводилось 128Кбайт. Это верный признак того, что фирма IBM намеривалась реализовать все эти возможности.

Это все, что следует знать читателю, прежде чем приступать к более глубокому изучению вопросов отображения информации на экране дисплея. В следующем параграфе мы изложим наши взгляды на принципы построения графических

систем, а затем рассмотрим детали построения хранимого образа экрана и обсудим такие понятия как "страницы" и "атрибуты".

Память дисплея физически размещается вместе с другими схемами дисплея на плате адаптера. Однако логически (для центрального процессора) она является частью основного адресного пространства компьютера. Для дисплея отводится 128Кбайт памяти в блоках A и B с адресами A0000(16) - BFFFF(16), однако оба стандартно поставляемых дисплейных адаптера используют лишь два небольших фрагмента этой области памяти. Одноцветный адаптер использует 4К дисплейной памяти, начиная с адреса B800. Оставшаяся память, в частности, 64-килобайтный блок от A000 до B000 отводится для развития возможностей дисплея, например, для использования дисплейного графического адаптера.

Одноцветный и цветной графические адаптеры используют для вывода информации на экран дисплея так называемый битовый образ изображения (экрана) хранящегося в описанной выше памяти. Каждая ячейка области памяти соответствует определенной позиции экрана, между ними установлено взаимно однозначное соответствие.

Схемы дисплея периодически считывают информацию из памяти и выводят ее на экран. Программа может непосредственно изменять содержимое памяти, тем самым изменяя содержимое экрана. Контроллер экрана служит для связи памяти дисплея с монитором, преобразуя поток получаемых из памяти битов в электронные пучки, направляемые в определенные точки экрана. Точки экрана называть пикселями, пэлами или элементами изображения. Они образуются при столкновении потока электронов с люминофором поверхности электронно-лучевой трубки. Поток электронов испускается электронной пушкой и построчно перемещается сверху вниз. По мере его движения контроллер электронно-лучевой трубки модулирует его интенсивность в зависимости от комбинации битов, полученных из памяти дисплея.

Видеосхема обновляет экран 60 раз в секунду, делая меняющееся изображение отчетливым и ясным. В конце каждого цикла обновления экрана электронный пучек должен переместиться из нижнего правого угла экрана в его верхний левый угол. Это перемещение называют вертикальным обратным ходом. Во время обратного хода луча его подача на поверхность экрана блокируется.

Время обратного вертикального хода луча составляет 1,25мсек и может учитываться программистом.

8.3. Структура копии изображения экрана

Выше уже говорилось о том, что дисплеи компьютера IBM/PC относятся к классу дисплеев, хранящих битовый образ экрана в памяти (карту) памяти. В связи с этим обстоятельством имеет смысл рассмотреть принципы построения этого образа.

Описываемый здесь образ экрана в памяти применим как к монохромному, так и к цветному графическрому адаптерам, работающим в тескстовом режиме. Графические режимы будут рассмотрены в следующей главе с учетом особенностей хранения изображений в памяти (Более подробные сведения о различных режимах работы цветного графического адаптера и некоторые интересные возможности описаны в разделе 8.11.).

Поскольку дисплейные адаптеры используют специальную двухвходовую память, не следует удивляться тому, что она размещается особым образом. Физически память дисплея размещается на схемной плате и плате расширения адаптера. Схемная плата содержит также микросхему контроллера дисплея и все вспомогательные элетронные схемы. Эта плата, подобно другим платам для IBM/PC, вставляется в один из дополнительных разъемов, предназначенных для поключения вспомогательного оборудования.

Несмотря на то, что память дисплея физически размещается на плате контроллера дисплея, логически она ничем не отличается от обычной памяти в том смысле, что допускает считывание и запись информации путем обращения по адресу. Дисплеям двух различных типов отводятся различные участки памяти (об этом уже говорилось при обсуждении основ использования памяти персонального компьютера в главе 3), поэтому они могут быть одновременно подключены к одному компьютеру. Монохромному дисплею отводится область памяти, начиная с шестнадцатеричного адреса B000, а цветному графическому - с адреса B800.

Читателю, знакомому с Бейсиком, напомним, что доступ к памяти осуществляется с помощью операторов PEEK и POKE после
того, как установлен нужный указатель сегмента

```
10 DEF SEG = &HB000 (память монохромного дисплея)
20  DEF  SEG  =  &HB800  (память  цветного  графического
дисплея)
```

После приведенных кратких сведений о принципах работы дисплея с хранимым образом экрана в памяти имеет смысл еще раз обратиться к диаграмме на рис. 8.1.

Процесс отображения содержимого экрана в память начинается с определения позиций отдельных символов экрана. Каждой позиции экрана соответствует два байта памяти. Совместно эти два байта задают собственно объект подлежащий, отображению на экране и то, как его следует отобразить.

Первый из двух байтов (имеющий четный адрес) специфицирует собственно объект (т.е. "что" отображается на экране). Это шестнадцатеричный код символа ASCII. Второй байт (имеющий нечетный адрес) специфицирует режим отображения первого байта (т.е. "как" символ отображается на экране). Это так называемый байт-атрибутов. В случае цветного графического адаптера байт-атрибутов управляет цветом. В случае монохромного адаптера байт-атрибутов управляет яркостью и подчеркиванием символа; это своего рода монохромные эквиваленты цвета. Кроме того байт-атрибутов устанавливает режим мерцания символа (для адаптеров обоих типов). Детальное описание байта-атрибутов приводится в разделах 8.5 и 8.6.

Первой позиции экрана (верхний левый угол соответствует таким образом два байта в памяти дисплея со смещением 0. Следующей позиции (смещенной на одно знакоместо вправо) соответствуют два байта памяти со смещением 2, Так продолжается до конца первой строки экрана дисплея. Для монохромного дисплея, а также для цветного графического дисплея, работающего в 80-ти позиционном режиме последнему символу строки соответствует пара байтов со смещением 158 (т.е. 79x2). Для цветного графического дисплея, работающего в 40- позиционном режиме, последней позиции первой строки соответствует пара байтов памяти со смещением 78 (т.е. 39x2).

При переходе к новой строке позиции экрана продолжают отображаться парой смежных байтов таким образом, что первой позиции следующей строки соответствует пара байтов, расположенных непосредственно вслед за байтами последней позиции предыдущей строки. Область памяти используется без промежутков, т.е. какие-либо пробеловы между концом одной строки и началом другой строки - отсутствуют. Разумеется, речь идет о физических промежутках, поскольку в логическом плане вполне естественно говорить о вычислении местоположения границы между строками.

Все сказанное выше распространяется как на 80-ти позиционный, так и на 40-позиционнный режимы. Поэтому при переходе из одного режима в другой должна измениться структура расположения в памяти позиций и строк.

Монохромный и цветной графический дисплеи в 80-ти позиционном режиме требуют 4000 байтов: 80 позиций x 2 байта x 25 строк. Цветной графический адаптер в 40- позиционном режиме требует вдвое меньшей памяти, т.е. порядка 2000 байт.

Следствием такой схемы хранения образа экрана в памяти является то, что отображаемые символы могут располагаться в любом другом месте памяти. А это, в свою очередь, создает неудобства при перемещении сообщений в память дисплея. Сообщения или любые другие строки отображаемых данных не могут быть перемещены по месту назначения за один шаг, если форма их хранения в программах не предполагает наличия байтов-атрибутов. Сообщение, таким образом, должно быть представлено в виде чередующейся последовательности содержательных и атрибутивных байтов. Поскольку большая часть данных выводятся на экран в виде строк символов с одними и теми же атрибутами, то возникает потребность в специальных программах перемещения данных в нужную область памяти, которые бы попутно дополняли их нужными атрибутами. Листинг 8.2 содержит тексты некоторых служебных программ на Паскале, реализующих эти функции; в параграфе 8.7. этот вопрос рассмотрен более детально.

Поскольку в памяти, отведенной дисплею, отсутствуют физические границы между строками, то могут быть автоматически перенесены в следующие строки, путем простого использования очередных адресов. Это довольно эффективный способ решения проблемы переноса сообщений с минимальными затратами. В тех случаях, когда важно фиксировать факт достижения сообщением конца строки, следует предусматривать в программах проверку и обработку этого условия. Для использования монохромного адаптера требуется 4000 байт памяти. Это приблизительно соответствует 4К (4096). Лишние 96 байтов не используются и остаются свободными. Цветной графический адаптер требует значительно большего объема памяти, а именно, 16К. Это объясняется большими потребностями памяти.

Объем памяти, используемой цветным графическим адаптером, составляет 16К байт. Такой значительный объем памяти определяется потребностями графических режимов, детально описываемых в следующей главе. Персональный компьютер использует излишки памяти (когда эта память реально не используется в графических режимах) путем организации (в текстовом режиме) набора копий кадров экрана. Эти копии принято называть страницами. В следующем параграфе мы рассмотрим принципы организации и функционирования страничного механизма.

8.4. Страничный механизм цветного графическоно дисплея

Материалы настоящего параграфа в первую очередь

относятся к компьютерам IBM/PC, снабженным цветным графическим адаптером. Однако, по мере изложения станет ясно, что эта концепция может быть распространена и на систему с монохромным адаптером. Поэтому, если читателя интересуют только системы второго класса (либо универсальные системы) данный параграф, несомненно, представляет интерес.

В отличие от монохромного адаптера, объем памяти которого в точности соответствует размеру заполненного кадра дисплея, цветной графический адаптер располагает памятью гораздо большего объема, чем это требуется для работы в текстовом режиме. Это объясняется тем, что для работы в графическом режиме требуются значительные ресурсы памяти. Фирма IBM пошла по пути реализации страничного механизма работы с кадрами изображения.

Страница представляет собой образ (копию) содержимого экрана дисплея в памяти компьютера. В 80-ти позиционном текстовом режиме цветной графический адаптер требует памяти в 4 раза (а в 40 - позиционном текстовом режиме в 8 раз) меньше фактически имеющейся. Поэтому память этого адаптера может хранить либо 4, либо 8 страниц.

Любая страница изображения представляет собой полную копию содержимого экрана дисплея, включающую байты данных, байты-атрибутов и построчно-логическое разбиение символов. Работая в 80-ти позиционном режиме, цветной графический адаптер может хранить четыре полных копии одного кадра экрана монохромного адаптера. При работе в 40 - позиционном режиме число таких кадров увеличивается до 8; при этом каждый кадр обладает вдвое меньшей информационной емкостью.

Информация, хранящаяся в каждой из четырех или восьми страниц, может в любой момент времени быть отображена на экране. Остальные страницы в это время, подобно актерам, ожидающим выхода на сцену, ждут своей очереди. Получив команду, дисплейный адаптер переключается с одной страницы на другую, таким образом мгновенно обновляя содержимое экрана.

Прользовательская программа может формировать данные, подлежащие отображению на дисплее, путем обычной записи по соответствующему адресу, либо путем использования функций обслуживания BIOS, позволяющих оперировать с нужной страницей. После того, как актуализация данных закончена, сформированный в памяти образ экрана может быть мгновенно отображен на дисплее путем переключения адаптером активной страницы. Такой подход делает программу более оперативной, поскольку пользователь не ощущает трудоемкого процесса формирования образа экрана - он видит лишь быстрое появление результата.

Возможности программ в части использования страничной

техники представляют довольно интересное явление. Если программа использует несколько стандартных форматов для представления информации, то она может сформировать каждый формат только один раз и сохранить его в страничной памяти дисплея. В случае возникновения потребности перехода от одного формата к другому программа может, сохранив текущую страницу на экране, форматировать данные другой страницы и завершив этот процесс, переключить адаптер дисплея на новую страницу. Такого рода техника работы с памятью дисплея создает впечатление, что программа обладает более высокой производительностью, поскольку изображение появляется мгновенно. В выигрыше оказывается и пользователь программы, получая более стабильное изображение на экране дисплея. Поскольку мелкие изменения не допускаются, то надежность человеко-машинного интерфейса повышается - в этом случае нет необходимости каждый раз отыскивать на экране места возможных изменений и проверять все ли на месте.

Еще одна потенциальная сфера использования страничной техники при работе с дисплеем состоит в более совершенной передаче динамических изображений. Если же изменения производятся "за кадром" и представляются пользователю в виде цельного изображения путем замены активной страницы, то динамический характер изображения передается более плавно.

Дисплейные страницы пронумерованы от 0 до 3, либо от 0 до 7 в случае более мелких 40 позиционных страниц. Нулевая страница, как это можно предположить, располагается в начале области памяти дисплея; вслед за ней располагаются остальные страницы. Началу каждой страницы соответствует адрес кратный 1К. Поясним это на примере. Дисплей, работающий в 80-ти позиционном режиме, требует для каждой страницы 4000 байт памяти (25 строк х 80 позиций х 2 байта на позицию). Первая страница, то есть страница, имеющая 0 номер, располагается в памяти цветного графического адаптера с 0 смещением (это соответствует параграфу с адресом B800). Вторая страница, то есть страница, имеющая номер 1-й, размещается не вплотную за первой страницей (с относительного адреса 4000), а с ближайшего адреса кратного одному килобайту (относительный адрес 4К или 4096 байт). В 80-ти позиционном режиме работы дисплея страницы размещаются с интервалом в 4К байтов, а в 40-позиционном - с интервалом в 2К байта. На рисунке 8.2. показано размещение страниц в памяти дисплея.

Управление страничным механизмом дисплея можно осуществлять в рамках Бэйсика. Пользователь, работающий на Паскале такой возможности лишен, если он не выходит за рамки этого языкового процессора; если же связь с BIOS-ПЗУ осуществляется через программу, написанную на ассемблере, то

эта задача вполне разрешима. В параграфе 8.11 мы покажем как это делается (готовая программа записана на дискету, прилагаемую к настоящей книге).

Монохромный дисплей не обладает описанными функциональными возможностями, однако эта концепция может быть реализована (имитирована) в обычной памяти. Пользовательские программы могут использовать эту память для хранения полного образа экрана, своего рода эквивалента дисплейных страниц. Эти пассивные экранные образы можно переместить в память дисплея используя средства строковой пересылки микропроцессора 8086/8088.

Пользователям, работающим на Паскале, предоставляется встроенная процедура MOVESL. Я использовал эту процедуру в ряде своих программ и могу засвидетельствовать, что загрузка экрана происходит совершенно незаметно для человеческого глаза - смена изображения производится мгновенно, как и в случае смены страницы в цветном графическом адаптере. Описанный метод весьма эффективен - настоятельно рекомендую его применять во всех программах, не предъявляющих особых требований к объему используемой памяти.

Причина, по которой лишь немногие программы используют страничный механизм цветного графического адаптера очевидна: большинство программ для IBM/PC должны работать одинаково хорошо как в случае цветного, так и в случае монохромного монитора.

Эффективное использование страничного механизма может быть достигнуто только в том случае, если программа специально написана для применения в составе системы обработки графических образов, либо когда программа настолько важна, что можно ожидать подстройку аппаратуры под нужды этой программы. К числу известных примеров относится программа Context MBA. Следует также иметь в виду, что имитация описанного выше страничного дисплейного механизма всегда осуществима для любого компьютера IBM/PC, имеющего достаточный объем памяти.

8.5. Атрибуты изображений

Следом за байтом символа, расположенного в памяти дисплея по четному адресу расположен байт атрибутов, описывающий особенности отображения на экране дисплея этого символа. В данном параграфе мы рассмотрим структуру байта атрибутов и назначение всех его компонентов.

Программа, написанная на Бэйсике, приведенная в приложении (листинг 8.1) демонстрирует все возможные

комбинации битов байта атрибутов как для монохромных, так и для цветных графических дисплеев. В процессе выполнения этой программы у пользователя формируется представление о функциональных возможностях и особенностях работы с атрибутами изображения.

Байт атрибутов управляет мерцанием символа и его цветом (или монохромным аналогом цвета). Этот байт состоит из восьми разрядов, причем каждому разряду отводится своя роль в процессе построения изображения. На рисунке 8.3 приведена схема, иллюстрирующая назначение каждого разряда. Существует множество различных способов идентификации отдельных разрядов
байта. Здесь мы будем придерживаться схемы нумерации этих разрядов слева-направо (см.рис. 8.3).

Все восемь разрядов байта разбиты на четыре класса и рассматриваются в контексте этого разбиения. Все, что выводится на экран - в конкретную его позицию - состоит из двух компонентов: собственно символа (очертания) и его окружения или фона. Поскольку цветной дисплей имеет три основных цвета (красный, зеленый и синий), то очертанию и фону соответствуют по три разряда байта-атрибутов, описывающие их цвета (или монохромные аналоги цвета). Таким образом: разряды с 6-го по 8-й управляют цветом символа, а разряды с 2-го по 4-й управляют цветом фона. Они и образуют два класса признаков (из четырех) байта атрибутов.

Оставшиеся два класса признаков фиксируют интенсивность и мерцание изображения. Под эти признаки отводится по одному разряду. Первый разряд каждого байта атрибутов описывает мерцание фоновой составляющей позиции экрана (если бит установлен в 1, то имеет место мерцание). Заметим здесь, что мерцание распространяется только на фоновую составляющую и не затрагивает собственно очертания символа.

Последний класс признаков (5-й разряд) байта атрибутов предназначен для управления интенсивностью фона. Если 5-й разряд установлен в 1, то в случае монохромного дисплея яркость символа будет повышена; в случае цветного дисплея будет иметь место более яркий и светлый тон. Некоторые цветные мониторы не используют бит интенсивности цвета; для этих мониторов яркие фоновые цвета совпадают с обычными.

Для получения разнообразных цветов соотвествтенно устанавливаются или сбрасываются разряды тех или иных цветов. Так, для получения красного фона следует устанавливать в"1" разряд красного цвета (6-й разряд), а два остальных разряда обнулить. Комбинация двух основных цветов дисплея позволяет синтезировать дополнительные цвета. Смесь зеленого и синего цветов дает сине-зеленый цвет, так называемый циан. Если все

три разряда находятся в нулевом состоянии, то в результате будет получен черный цвет. Смесь всех трех основных цветов дает на выходе белый цвет. При внимательном рассмотрении символов белого цвета на цветном экране можно различить все три основных цвета.

Порядковый номер разряда	Значение *	Функция
1-й	128	Мерцание
2-й	64	Красная составляющая ! Фоновый цвет
3-й	32	Зеленая составляющая !
4-й	16	Синяя составляющая !
5-й	8	Интенсивность! Фоновый цвет
6-й	4	Красный ! Цвет
7-й	2	Зеленый ! очертания
8-й	1	Синий ! (символа)

*)
Численное значение позиции разряда. Используется в случае задания атрибута в виде числа. Например: Мерцание (128) зеленого символа (2) на красном (64) фоне - 128 + 2 + 64 = 194

Рис.8.3. Управляющие атрибуты дисплея

Полный перечень всех возможных цветовых сочетаний будет приведен в следующем параграфе . По мере рассмотрения различных аспектов, связанных с использованием атрибутов изображения мы узнаем много нового, в частности, специфику их работы как в случае цветного графического, так и в случае монохромного адаптера.

Операционная система DOS обеспечивает нормальный режим работы с белыми символами на черном фоне даже при наличии цветного экрана видеомонитора. Большинство программ, работающих под управлением DOS, генерируют выходные данные в
черно-белом варианте. Однако при наличии программы управления атрибутами изображения (например, программ Reverse и Screen-Atribute из библиотеки "Нортон Ютилити") можно привести их к цветному варианту. При работе с цветным монитором я нахожу, что ярко-желтые символы на синем фоне гораэдо меньше утомляют глаза, чем любое другое сочетание цветов. Поэтому для себя устанавливаю следующую комбинацию атрибутов: 00011110 (или 1E в шестнадцатеричной системе счисления). Воспользовавшись программой, написанной на языке Бэйсик (см. листинг 8.1), Вы имеете возможность сопоставить между собой все допустимые сочетания цвета очертания символа и его фона и после этого остановить свой выбор на одном из них.

Фирма IBM выработала оригинальный подход к решению проблемы совместимости монохромного и цветного дисплеев. Первое, что было сделано - осуществлен выбор основных технических решений, касающихся цветного дисплея (наличие трех базовых составляющих цвета для очертания символа и его фона, а также мерцание и высокая интенсивность); после этого средствами монохромного дисплея удалось добиться весьма эффективной интепретации цветовых составляющих.

Атрибуты монохромного дисплея имеют ряд положительных качеств, им присущи однако и недостатки, связанные с обеспечением целостности и совместимости. Проиллюстрируем это положение сначала более простыми примерами, а затем перейдем к более сложным случаям.

Разряд мерцания (разряд 1), как и разряд яркости, называемый также разрядом повышенной интенсивности свечения (разряд 5), используется одинаково как монохромным, так и цветным графическим дисплеем. Если разряд мерцания установлен в единичное состояние, то отображаемый символ мерцает, если в единичном состоянии находится разряд яркости, то отображаемый символ выделяется повышенной яркостью.

Оставшиеся разряды байта атрибутов специфицируют цвета для очертания и фона символа. Заметим здесь, что для монохромного дисплея понятие цвета отсутствует. Последнее утверждение не следует понимать буквально, поскольку, строго говоря, монохромный дисплей имеет два цвета: зеленовато-фосфорный и черный. Зеленовато-фосфорный (белый)
цвет соответствует свечению люминофора, а черный цвет

свидетельствует об отсутствии свечения. Любая возможная комбинация цветов очертания и фона в случае монохромного дисплея изображена на экране в виде зеленоватого символа на черном фоне.

Стандартный набор признаков в случае монохромного дисплея обеспечивает режим отображения такой же как и в случае цветного графического дисплея, а именно: белые символы (кодовая комбинация разрядов цветности 111) на черном фоне (кодовая комбинация разрядов цветности 000); исключение составляют три особых случая. Любой код цветности (за исключением трех случаев рассматриваемых ниже) вызывает последствия, аналогичные стандартному набору признаков, т.е. свечение зеленоватых символов на черном фоне.

Два из трех особых случаев, представляются более или менее очевидными. Если кодовая комбинация признаков соответствует режиму отображения "черное-на черном" (т.е. все разряды цветности находятся в нулевом состоянии), то любой отображаемый символ становится невидимым. Если же кодовая комбинация признаков соответствует режиму отображения "черное-на-белом" (разряды очертания хранят код 000, а фоновые разряды - код 111), то имеет место так называемое обратное или реверсное изображение (черные символы на зеленовато-фосфорном фоне). Здесь имеется полная аналогия с цветным дисплеем, когда черный символ на черном фоне неразличим, а изображение черного символа на белом фоне называется обратным. Вполне естественно ожидать, что четвертая комбинация признаков, соответствующая режиму белый символ на белом фоне, порождает невидимые символы. Однако, фирма IBM пошла другим путем. В случае монохромного дисплея комбинация признаков байта атрибутов, соответствующая режиму белый символ на белом фоне, фактически сводится к использованию стандартного формата изображения зеленоватых символов на черном фоне (аналогично тому, как то происходит в случае всех остальных комбинаций цветовых признаков).

И последний аспект представления информации на экране монохромного дисплея связан с возможностью подчеркивания. Фирма IBM предлагает своим пользователм монохромный дисплей, обладающий возможностью подчеркивания символов. В терминах признаков цветности подчеркивание представляется в виде синего очертания (код цветности 001) и черного фона (код цветности 000). Собственно цветной графический адаптер не предусматривает реализации функции подчеркивания, однако сам факт использования синих символов на черном фоне создает впечатление некоторого акцентирования.

Поскольку монохромный дисплей допускает представление обратных изображений и реализует функцию подчеркивания,

можно ожидать, что допустимо их сочетание в рамках одного кадра. Однако это не так. Цветовые комбинации, которые можно было бы здесь использовать (синий символ на белом фоне или черный символ на синем фоне), интерпретируется системой как стандартные, ничем не отличающиеся от других цветовых комбинаций.

8.6. Особенности воспроизведения цвета

В данном параграфе мы рассмотрим различные особенности воспроизведения цвета на цветном видеомониторе, снабженном цветным графическим адаптером. Следует иметь в виду, что черно-белый монитор с цветным графическим адаптером цвет не воспроизводит. Здесь следует также отметить, что многие широко распространенные черно-белые мониторы не корректно воспринимают сигналы цветного изображения, поэтому информация, содержащая цветовые компоненты и выводимая на черно-белый монитор, может оказаться искаженной или невидимой.

Теоретически цветной дисплей компьютера IBM/PC воспроизводит шестнадцать различных цветов. Это число получается следующим образом. Во-первых, имеется восемь различных цветовых сочетаний, образуемых тремя основными цветами - красным, зеленым и синим. Во-вторых, любой цвет очертания символа может быть воспроизведен в двух режимах; в режиме нормальной и в режиме повышенной яркости (цвет высокой интенсивности). Управление этими режимами осуществляется с помощью бита интенсивности. Таким образом, допускается воспроизведение шестнадцати цветов при отображении очертания символа и восьми цветов при отображении фона.

(Графический режим, который будет рассмотрен в следующей главе также допускает использование шестнадцати цветов, однако особенности их использования для очертания символа и его фона существенно отличаются).

Все шестнадцать цветов на практике никогда не используются. Один из восьми основных цветов - черный; в связи с этим неясно, что такое черный цвет повышенной яркости. Фирма IBM называет этот цвет темно серым. В принципе этот цвет можно воспроизвести на экране цветного дисплея, однако делается это весьма редко. Таким образом, количество цветов уменьшается до пятнадцати. Если же учитывать, что не все цветные мониторы используют бит яркости, то количество воспроизводимых цветов ограничивается только основными. Фактически это число еще меньше - только семь - поскольку черный цвет используется редко.

Ниже приводятся все шестнадцать комбинаций цветов,

обеспечиваемых 4-мя разрядами, включающими разряд яркости или повышенной интенсивности свечения.

Яркость	Красный	Зеленый	Синий	Номер	Цвет	Примечание
0	0	0	0	0	Черный	Изображение отсутствует
0	0	0	1	1	Синий	
0	0	1	0	2	Зеленый	
0	0	1	1	3	Циан	Сине-зеленый
0	1	0	0	4	Красный	
0	1	0	1	5	Пурпур-ный	Светло-пурпурный, состоящий из красного и синего
0	1	1	0	6	Корич-невый	Темно-желтый; для большинства дисплеев - желтый
0	1	1	1	7	Светло-серый	Обычный белый
1	0	0	0	8	Темно-серый	Черный, повышенной яркости (!?)
1	0	0	1	9	Светло-синий	
1	0	1	0	10	Светло-зеленый	
1	0	1	1	11	Циан светлый	Сине-зеленый, светлый
1	1	0	0	12	Светло-красный	
1	1	0	1	13	Светло пурпур-ный	

1	1	1	0	14	Желтый !Светло- !желтый
1	1	1	1	15	Белый !Белый, !повышенной !яркости

В столбце "номер" указывается числовое обозначение цвета, используемое в языке Бейсик. Числа от 16 до 31 служат для обозначения тех же цветов, что и числа от 0 до 15, число 16 добавляется для установки атрибута мерцания.

Для формирования у пользователя более строгого представления об использовании цвета в программах, приведем здесь краткие сведения о цветовых возможностях видеоминитора.

Каждый пользователь должен отдавать себе отчет в особенностях воспроизведения цвета повышенной интенсивности. Такие цвета как циан и пурпурный зрительно могут восприниматься как более интенсивные, но поскольку они вдвое ярче основных цветов (красного, зеленого и синего), то они воспринимаются сравнительно блеклыми. Напомним здесь, что яркость - это количество испускаемого света. Количество света еще не говорит о насыщенности цвета - чаще всего это говорит о размытости цвета. В то же время в отношении желтого цвета справедливо следующее утверждение - чем ярче цвет, тем выше его насыщенность (интенсивность). Желтый цвет, подобно красному, зеленому и синему представляет особый цвет.

Комфортность восприятия цвета меняется в зависимости от качества экрана дисплея и от специфики реакции глаза. Большинство пользователей, как нам представляется, считают, что синий, красный, желтый и, возможно, черный относятся к ортогональным цветам, обеспечивающим наибольшую комфортность
восприятия. Следующими (на шкале комфортности восприятия) цветами являются зеленый и пурпурный. И последними в списке наиболее комфортных цветов являются циан и белый.

Каждый, кто имеет серьезные намерения в отношении цветовых возможностей IBM/PC должен довольно внимательно отнестись к выбору цвета и (по возможности) проверить свои решения на нескольких экранах различных типов. Неверный выбор может осложнить восприятие информации с экрана дисплея.

8.7. Режим прямого управления видеомонитором

Существует два основных способа отображения информации

на экране компьютера IBM/PC.

Первый способ, который я буду называть косвенным, состоит в использовании копии изображения, хранящейся в памяти дисплея. Фирма IBM, однако относится к этому способу неодобрительно. В первой редакции руководства по программированию этот способ рассматривается как образец "дурного тона" в программировании, поскольку программы становятся зависимыми от аппаратных особенностей IBM/PC. В случае замены типа дисплея (что является вполне реальным,(см. параграф 8.2) или какой-либо иной аппаратной модификации пользовательские программы подлежат модификации.

Наиболее эффективный способ генерации изображения на экране дисплея состоит в использовании функций обслуживания BIOS; в этом случае при любых модификациях компьютера (либо для последующих моделей IBM/PC) фирмы-изготовители позаботятся о внесении соответствующих изменений в BIOS с тем, чтобы пользовательские программы остались неизменными. Следует однако отметить, что функции обслуживания BIOS-ПЗУ, ориентированные на работу с дисплеями, не вполне корректны; я даже считаю их малопригодными. Во многом это объясняется тем, что любая мало-мальски серьезная программа обходит их и работает непосредственно с памятью дисплея, хранящей изображение. В отличие от точки зрения фирмы IBM на "дурной тон" в программировании, я считаю что использование памяти дисплея, хранящей кодированную копию изображения, вполне обоснованным решением в случае потребности получения быстрых и сложных изображений. Генерация изображения на экране дисплея с помощью адекватного образа памяти обладает огромными преимуществами. Изображение формируется быстро, непосредственно и весьма эффективно. Процесс генерации протекает достаточно просто. Программа, приведенная в приложении 8.1, иллюстрирует особенности реализации этого процесса на языке Бейсик, а текст программы приложения 8.2 содержит ряд процедур обслуживания, предназначенных для прямого обмена с памятью дисплея программ, написанных на языке Паскаль. Эти процедуры могут использоваться в качестве базиса для любых программ обмена с дисплеем, написанных на Паскале, или большинстве других языков.

Принцип работы таких программ весьма прост - во-первых, программе необходимо сообщить тип дисплея - монохромный или цветной. Это можно сделать двумя способами: либо путем опроса пользователя (программа в процессе своей настройки задает пользователю прямой вопрос), либо путем опроса операционной системы (программа в этом случае получает доступ к внутренним таблицам операционной системы, содержащим сведения о типах устройств).

Используя функцию обслуживания BIOS-ПЗУ (рассматривается в разделе 8.11), можно получить информацию о типе используемого дисплея. Это наиболее эффективный способ настройки программ.

После определения типа дисплея необходимо сформировать ссылку (указатель) на соответствующий участок памяти; в случае монохромного адаптера это адрес шестнадцатеричного параграфа B000, а случае цветного графического адаптера - B800. В рамках языка Бейсика это можно осуществить с помощью оператора DEF SEG.B Паскале это делается с помощью сегментированных типов. Любой из названных способов осуществим весьма просто.

Поскольку пользователь имеет возможность обращаться к памяти дисплея, то тем самым у него появляется возможность с одной стороны отображать на экране нужную ему информацию, а с другой стороны он имеет возможность считать то, что уже отображено.

Единственное возникающее при этом осложнение - относительная адресация. Пользователь обязан учитывать наличие байтов-атрибутов (размещаются по нечетным адресам); если необходимо принимать в расчет разбивку экрана на строки, то программы должны выполнить соответствующие вычисления. Если используется страничный механизм дисплея, то следует так же учитывать особенности размещения страниц его адреса кратным 2К и 4К.

В приложении 8.2 приведены наиболее распространенные модули, требуемые для программирования на Паскале. Если возникает потребность в дополнительных модулях, то они могут быть легко добавлены к существующим по уже известной схеме (см. приложение 8.2).

8.8. Управление курсором

Курсор представляет собой важнейший компонент видеодисплея. Обычно он используется в качестве индикатора местоположения символа, вводимого с клавиатуры, однако он используется также и в качестве указателя на объекты, расположенные на экране. В данном параграфе мы рассмотрим принцип управления курсором. Первое, о чем мне хотелось бы здесь сказать, связано с тем, что мерцающий курсор IBM/PC может оказаться неприемлемым для многих пользователей.

Рассмотрим некоторые альтернативные варианты.

В связи с использованием мерцающего курсора возникает ряд проблем чисто зрительного восприятия. Компьютер IBM/PC имеет курсор стандартной формы, появляющиеся на экране при

включении компьютера в электрическую сеть; это небольшое мерцающее пятно, расположенное ниже позиции символа. Некоторые считают, что такой курсор трудно отыскать на экране, заполненном информацией .

Существует возможность увеличить размер курсора, доведя его до размера символа. В качестве образца процедуры увеличения размера курсора до половины размера символа следует запустить интерпретатор с языка Бейсик и нажать клавишу Insert (Вставка). С помощью оператора LOCATE можно установить любой размер курсора. Но при этом оказывается, что большой мерцающий курсор действует на глаз утомляюще, особенно при длительной работе.

Мерцающий курсор можно использовать двумя способами. Первый способ связан с использованием атрибута обратного изображения (шестнадцатеричный код 70) в отношении той позиции экрана, в которой находится курсор. В этом случае генерируется большое световое пятно, на фоне которого виден символ; таким образом, возникает сплошной (имеющий размер символа) немерцающий курсор. Курсор такого типа хорошо воспринимается в зрительном плане и не действует утомляюще на глаз. В тех случаях, когда Бейсик работает с цветным графическим монитором в графическом режиме, оказывается, что курсор генерируется сходным образом. Объясняется это тем, что графический режим не предусматривает аппаратной реализации мерцающего курсора, поэтому Бейсик вынужден имитировать курсор. Но даже в тех случаях, когда предусмотрен мерцающий курсор, реверсивное изображение считается наилучшим способом представления курсора (здесь уместно заметить, что текстовый редактор, использованный при подготовке рукописи этой книги (Vedit), имеет именно такой курсор).

Второй способ использования курсора особенно уместен, когда программа выводит на экран перечни (списки) каких-либо информационных объектов. Назначение курсора состоит не только в фиксации на экране позиции табуляции (позиции размещения символа), но также и в реализации процедуры выбора. Последнее характерно для программ обработки меню, списков файлов и вообще наборов альтернативных вариантов. Для таких случаев я бы предложил использовать два символа "стрелки", именно CHR$(16) для стрелки, направленной вправо, и CHR$(17) для стрелки, направленной влево. (Входящая в состав пакета программ, прилагаемого к данной книге, программа DiskLook, а также многие другие программы из числа "Нортон Ютилити" используют правую стрелку).

Если задача генерации искусственного курсора не стоит, то задача управления курсором практически возникает всегда. Впрочем, в случае искусственного создания курсора может

возникнуть потребность вообще убрать его с экрана. Курсор обладает двумя характеристиками, подлежащими управлению: размером и позицией.

В случае аппаратно-генерируемого курсора он представляется в виде совокупности мерцающих строк из числа строк развертки, генерирующих отображаемый символ. Поскольку строки, из которых он состоит, не могут управляться независимо, то у пользователя имеется возможность управлять лишь группой мерцающих строк. Монохромный дисплей имеет 13 строк развертки, а цветной графический - 7. Принято нумеровать строки сверху, начиная с 0 и вплоть до 13 (для монохромного дисплея) или до 7 (для цветного дисплея). Стандартное положение курсора (в момент запуска IBM/PC) - 12 и 13-я строки для монохромного дисплея и 7-я строка для цветного графического дисплея.

Пользователь имеет возможность задать положение курсора путем выделения произвольного количества этих строк, указав начальный и конечный номер строк развертки. Курсор, таким образом, может быть полноразмерным или занимать верхнюю, среднюю и нижнюю часть позиции символа.

Если номер начальной строки больше номера конечной строки (например: начальная строка имеет номер 12, а конечная строка номер 2), то курсор будет образован следующим образом: используются строки развертки, начинающиеся с начальной строки вплоть до конечной строки, а затем берутся самые верхние строки, вплоть до конечной строки. Таким образом, генерируется курсор весьма причудливой формы, состоящий из двух частей - верхней и нижней. Это весьма необычная конструкция, и у Вас может возникнуть желание посмотреть на нее; я бы, однако, рекомендовал хорошо подумать, прежде чем предлагать такое решение пользователям Ваших программ.

При необходимости генерации составного курсора следует запустить интерпретатор языка Бейсик и ввести команду

```
LOCATE ,,, 6,2
```

В результате на экране появится составной курсор и могут быть выполнены действия по модификации его размеров. Вторым управляемым параметром курсора является его местоположение. Курсор может быть расположен в любой строке или столбце экрана дисплея, кроме того, его можно скрыть, поместив в несуществующую строку 26. С помощью оператора LOCATE (язык Бейсик) можно управлять обоими параметрами. Для остальных языков это не так. Многие пользователи языка Паскаль считают, что отсутствие средств управления курсором является серьезным недостатком данной версии языка.

На дискете, прилагаемой к данной книге, содержатся процедуры, написанные на языке ассемблера; с помощью этих процедур может быть осуществлен доступ к курсору или реализовано управление им в рамках языка Паскаль или любого другого языка. Особенности реализации этих процедур описаны в параграфе 8.11; там же приведены и другие сведения, касающиеся обращения к BIOS-ПЗУ для видеодоступа.

8.9. Стандартный режим управления видеомонитором

Стандартный (или штатный) режим доступа к информации предполагает, что генерация изображений осуществляется без модификации содержимого участка памяти, хранящего закодированный образ экрана. В параграфе 8.11 будет рассмотрена возможность такого доступа, используя функции обслуживания BIOS-ПЗУ. В связи с этим мне все-таки неясно, почему все стремятся использовать именно функции обслуживания (исключение составляют пользователи, работающие на ассемблере), несмотря на то, что средства вывода языков высокого уровня (Паскаль и другие) обладают явными преимуществами.

В случае использования средств вывода языков высокого уровня (Паскаль и другие) возникает одна особенность. Вывод информации всегда осуществляется с учетом положения курсора. Поэтому, если производится манипулирование курсором с помощью функций обслуживания BIOS-ПЗУ, описываемых в параграфе 8.11, то это вполне согласится со спецификой работы средств вывода языков высокого уровня.

Таким образом, всегда следует придерживаться разумного сочетания двух основополагающих принципов работы: использование средств вывода базового языка программирования и достижения на этой основе высокого уровня контроля отображаемых данных и отказ от непосредственного использования кодированного образа экрана в памяти дисплея.

Придерживаясь этих концепций, можно достичь высокого уровня контроля за отображаемой информацией (в первую очередь это относится к атрибутам отображения); это может служить хорошей основой для построения высокомобильных систем программного обеспечения для персональных компьютеров.

8.10. Псевдографическиц режим

Символьная графика, или псевдографика представляет собой интересную и эффективную альтернативу использованию полномаштабных графических возможностей цветного графического дисплея.Программы, написанные в расчете на использование псевдографических возможностей имеют важное преимущество, они не зависят от типа используемого дисплея (черно-белый или цветной графический).

Под псевдографическими изображениями понимаются такие изображения, которые создаются на базе расширенного символьного набора кода ASCII. Следом за стандартными символами ASCII (первые 128 символов) расположены еще 128 символов, имеющих особую конфигурацию. Примерно половина этих
символов предназначена для представления чертежей и рисунков.

Прежде всего речь идет о прямых. В ASCII предусмотрен полный набор символов, позволяющих вычерчивать контуры прямоугольников, диаграмм и схем одиночными и двойными линиями. Многие программы для IBM/PC, включая демонстрационные программы, написанные на Бейсике и поставляемые в составе DOS, используют символы прямых, поэтому многим они знакомы.

8.11 Средства управления видеотерминалом уровня BIOS-ПЗУ

Ниже описываются функции обслуживания видеомонитора, реализуемые средствами BIOS-ПЗУ. Ряд этих функций относится к графическим режимам, которые будут детально рассмотрены в следующей главе; здесь мы опишем их лишь вкратце.

Существует категория читателей, которые непосредственно не будет использовать функции обслуживания, им важно понимать принципы и возможности системы.

BIOS-ПЗУ предлагает пользователю шестнадцать различных функций обслуживания для видеомониторов. Доступ к этим функциям осуществляется через прерывание с номером 16(10/16). В данном параграфе мы рассмотрим все эти прерывания и осмыслим их назначение. Программы 8.101 и 8.102, расположенные на дискете, прилагаемой к данной книге, представляют собой процедуры, написанные на языке ассемблера (процедуры первого уровня), для обращения к функциям обслуживания видеодисплея, реализуемым с помощью BIOS-ПЗУ; кроме того, они содержат определения Паскаля и процедуры второго уровня, предназначенные для облегчения использования первых.

Первая процедура обслуживания видеомонитора имеет код

обслуживания 0, она используется для фиксации режима работы видеомонитора. Возможны восемь различных режимов работы видеомонитора, причем семь режимов ориентированы на цветной графический дисплей, а один на монохромный. Ниже приводится список режимов работы (в дальнейшем мы прокомментируем позиции этого списка).

Код	Режим
0	Текстовый, 40-позиционный, черно-белый (цвет подавлен) режим для цветного графического дисплея
1	Текстовый, 40-позиционный, цветной режим для цветного графического дисплея
2	Текстовый, 80-позиционный, черно-белый (цвет подавлен) режим для цветного графического дисплея
3	Текстовый, 80-позиционный, цветной режим для цветного графического дисплея
4	Графический, 320 x 200 пикселей, цветной режим для цветного графического дисплея
6	Графический, 640 x 200 пикселей, черно-белый режим для цветного графического дисплея
7	Режим монохромного дисплея

При наличии цветного графического адаптера с помощью указанных функций обслуживания довольно просто обеспечивается переключение режима. Можно было бы предположить, что при наличии двух адаптеров эта сервисная функция обеспечит переключение одного активного дисплея на другой, но, к сожалению, это не так. В BIOS-ПЗУ предусмотрен специальный признак, который отмечает факт наличия монохромного адаптера; если монохромный адаптер подключен, то BIOS блокирует любой запрос, связанный с переключением в режим цветного графического дисплея. Система, включающая в свой состав два видеомонитора может переключаться с одного дисплея на второй лишь с помощью специальных кодовых комбинаций (эти сведения только недавно опубликованы фирмой IBM). Что же касается одновременной активизации обоих дисплеев, то это, насколько

мне известно, попросту невозможно.

Черно-белые текстовые режимы (коды 0 и 2) работают аналогично соответствующим цветным режимам, но с блокировкой цветности. Смысл, вкладываемый в термин "пиксель", а также сведения о разрешающей способности графических режимов будут приведены в следующей главе.

Вторая функция обслуживания (код 1) используется для установки размера курсора. Как уже указывалось ранее - конфигурация и размер корсора фиксируются путем определения начальной и конечной строк развертки позиции курсора. Верхняя строка развертки имеет номер 0, а нижняя строка развертки - номер 7 (для цветного графического режима) или номер 13 (для монохромного режима). Если номер начальной строки больше номера конечной строки, то генерируется (двух)составной курсор. Особенности управления курсором описаны в параграфе 8.8.

Третья функция обслуживания (код 2) используется для перемещения курсора. Положение курсора описывается тремя параметрами - строка, позиция (столбец) и страница. Отсчет строк и позиций начинается с первого знакоместа (верхний левый угол экрана), имеющего номер 0. Если возникает необходимость использовать нумерацию от 1 до 25 (вместо нумерации от 0 до 24), то пользовательская программа должна выполнить пересчет номеров; программы обслуживания на Паскале, приведенные в приложении 8.102, реализуют именно эту функцию. Номер страницы (памяти дисплея) используется только в случае текстового режима работы цветного графического адаптера; при работе с монохромным дисплеем, а также в графических режимах номер страницы должен быть установлен в 0. Курсор можно вывести вообще за пределы экрана и, таким образом, сделать его невидимым. Для этого я бы рекомендовал использовать первую позицию строки, выходящую за пределы максимально возможной (если, например, нумерация строк и позиций ведется с 1, то следует указать первую позицию 26-й строки).

Положение курсора фиксируется путем занесения номера строки в регистр DH, а номера позиции (столбца) в регистр DL. Нумерация строк позиций начинается с нулевых координат (0,0) верхнего левого угла экрана. В графических режимах положение курсора также описывается в терминах координат "строка-позиция"; координаты пикселя не используются. Для нумерации страниц используются числа 0 - 7 (40-позиционный режим) или 0 - 3 (80 позиционный режим). Для графических режимов номер страницы должен быть равным 0.

Ниже приведены обобщенные сведения о содержимом регистров при обращении к функции обслуживания 2.

Номер функции обслуживания (регистры)	Параметры (регистры)
AH= 2	DH = номер строки DL = номер позиции BH = номер страницы (для графических режимов равен 0)

Здесь уместно сделать одно важное замечание, касающееся работы в цветном графическом многостраничном режиме. Если возникает потребность в перемещении курсора, то следует специфицировать страницу - таким образом исключается возможность поместить курсор в текущую активную страницу дисплея; любая другая страница в этом смысле является доступной. Каждая страница имеет собственное логическое местоположение курсора. Некоторые функции обслуживания (из числа перечисляемых ниже), применимые к любой странице изображения , действуют относительно положения курсора. Положение курсора фиксируется для каждой страницы изображения.

Четвертая функция обслуживания (имеющая код 3) предназначена для считывания положения курсора и его размера. В каком-то смысле она реализует процедуру, обратную процедуре, реализуемой совместным использованием двух предшествующих функций обслуживания. Здесь также следует специфицировать страницу изображения; для монохромного дисплея, а также для графических режимов ее номер должен равняться 0. В результате выполнения функции пользователь получает номера строки и позиции курсора, а также номера начальной и конечной строки развертки, формирующие курсор. Ниже приводятся сведения о составе и содержимом регистров.

1 Номер начальной строки должен быть загружен в регистр CH, а номер конечной строки - в регистр CL. По умолчанию для монохромного адаптера устанавливаются следующие значения CH =
12, CL = 13.

Здесь уместно заметить, что для номеров строк отведено только три байта (0 - 2) в указанных регистрах. Если пятый

бит регистра CH установлен в состояние "1" (т.е. в регистр загружено значение 32), то курсор исчезает. При переходе в графический режим этот бит автоматически устанавливается равным 1, с тем чтобы курсор не мешал графическому выводу. Этот прием также можно использовать для искусственного удаления курсора. Поскольку в графических режимах операции с курсором аппаратно не поддерживаются, то речь может идти о его имитации с помощью символа CH$(223) и изменения атрибутов фона.

Номер функции обслуживания	Параметры
AH= 1	CH = начальная строка курсора CL = конечная строка курсора
AH= 3	BH = номер страницы (для графических режимов устанавливается равным 0) DH = номер строки DL = номер позиции (столбца) CH = ночальная строка развертки курсора CL = конечная строка развертки курсора

Пятая функция обслуживания (функция с кодом 4) предназначена для считывания местоположения, указанного световым пером (для систем, имеющих световое перо). В результате выполнения фиксируется факт подключения светового пера; если световое перо подключено, то фиксируется также положение указанной точки. Положение задается как в терминах символов, так и в терминах пикселей (этот вопрос подробно обсуждается в следующей главе, посвященной графике).

Регистр AH предназначен для индикации подключения светового пера: если AH=1, то световое перо поключено, в противном случае (AH=0), световое перо не используется. Положение объекта, вычлененного с помощью светового пера формируется либо в виде содержимого регистров CH, BX

(положение пикселя). Поскольку величина, соответствующая (горизонтальной) позиции пикселя может превышать 255, то для нее отводится полное слово (регистр BX).

Ниже приводятся сведения о составе и содержимом регистров.

Номер функции обслуживания	Параметры
AH = 4	DH = номер строки символов DL = номер позиции (столбца)символа CH = номер строки пикселей (0 - 199) BX = номер позиции (столбца) пикселя

Световое перо не относится к числу наиболее распространенных аппаратных средств для IBM/PC. Это своего рода сдерживающий фактор, поскольку может использоваться только с дисплеями, обладающими очень малым временем послесвечения люминофора, т.е. такими дисплея, у которых светимость точки быстро падает после перемещения луча развертки в следующую точку экрана. Такие дисплеи быстро утомляют оператора. Поэтому хорошие дисплеи не нуждаются в световом пере. Для монохромного дисплея фирмы IBM применяется фосфорный дисплей с длительным послесвечением, облегчающий восприятие изображений. Можно считать, что световое перо получит ограниченное распространение (в основном для специальных приложений) в сфере персональных компьютеров.

Шестая функция обслуживания (функция с кодом 5) используется для установки (подключения) активной страницы в многостраничном текстовом режиме цветного графического дисплея. Номер страницы задается в регистре AL. Для 40-позиционных режимов номера страницы лежат в диапазоне 0 - 7, а для 80 позиционных режимов - в диапазоне 0 - 4. По умолчанию номер страницы принимается равным 0. Эта страница помещается в самом начале памяти дисплея, а каждая последующая страница с интервалом в 2K (для 40 позиционных

режимов) или через 4К (для 80 позиционных режимов). Большему номеру страницы соответствуют ячейки памяти с большими адресами.

Номер функции обслуживания	Параметры
AH = 5	AL = номер новой страницы дисплея (0 - 3 для режимов 2 и 3, 0 - 7 для режимов 0 и 1)

Две следующих функции обслуживания /функции с кодами 6 и 7/ используются для реализации режима "прокрутки" (скроллинга) информации на экране дисплея. Использование компьютера IBM/PC в режиме "прокрутки" представляет собой одну из интереснейших возможностей отображения информации, однако лишь немногие программы, используюь эту возможность. В режиме "прокрутки" можно перемесить содержимое экрана вверх или сниз на любое число строк. При этом отображаемые символы смещаются за пределы экрана вверх или вниз, а освободившаяся часть экрана заполняется пустыми строками. Особенность этого режима отображения состоит в том, что в роли экрана может выступать произвольная прямоугольная область экрана (окно), на которую распространяется "прокрутка"; остальные участки экрана остаются неизменными. Выделив несколько окон, можно организовать в каждом из них режим "прокрутки". Возможности обработки данных здесь огромные, остается лишь недоумевать, почему большинство программ их редко используют.

Функция с кодом 6 предназначена для "прокрутки" вверх, а функция с кодом 7 предназначена для "прокрутки" вниз. И в том, и в другом случае следует специфицировать два противоположных угла окна - верхний левый и нижний правый. Спецификация задается в терминавх строк и столбцов. Следует также задать число перемещаемых строк; это число может совпадать с размером окна. И последний об'ект спецификации - атрибут отображения строк заполнителей. Таким образом, имеется возможность управлять цветом окна. Если планируется использование техники работы с окнами, то имеет смысл выделить окно путем приписывания ему атрибута отображения, отличного от атрибута отображения всего остального экрана. С

помощью механизма прокрутки текст сообщения не может быть помещен в новые строки окна - для этого следует принять специальные меры.

Число перемещаемых строк указывается в регистре AL. Если AL=0, то все окно заполняется пустыми строками (то же самое происходит, если число перемещаемых строк превышает размеры окна). Положение и размер окна указываются в регистрах CX и DX: в CH указывается верхняя строка, а в DH - нижняя; в CL указывается левая позиция (столбец, а в DL - правый. Атрибуты изображения пустых строк указываются в регистре BH. Ниже приводятся общие сведения о составе и содержимом регистров.

Номер функции обслуживания	Параметры
AH = 6	AL = количество сдвигаемых строк CH = номер строки верхнего левого угла окна CL = номер позиции (столбца) верхнего левого угла окна DH = номер строки нижнего правого угла окна DL = номер позиции (столбца) нижнего правого угла окна BH = атрибуты изображения пустых строк

"Прокрутка" содержимого окна как правило выполняется в два этапа. Как только новая строка подготовлена к записи в окно, осуществляется вызов функции обслуживания с кодом 6 (или 7) для перемещения текущего содержимого окна. После этого в новую строку заносится нужная информация. Для этого используются функции обслуживания "установка курсора" и "запись символа". Ниже приводится пример, иллюстрирующий излагаемые положения.

```
DEBUG; вызов программы-отладчика DEBUG
  A
INT 10; вызов  Ассемблеоа
[Return]; выход из Ассемблера
```

R AX; извлечение содержимого регистра AX для просмотра и изменения
06 03; установка функции обслуживания с кодом 6, количество сдвигаемых строк 3
R CX; извлечение содержимого регистра CX для просмотра и изменения
050A; установка верхнего левого угла: строка 5, столбец 10
R DX; извлечение содержимого регистра DX для просмотра и изменения
1020; установка правого нижнего угла: строка 16, столбец 32
D 0 L 180; заполнение экрана
G 100 102; выполнение прерывания 10, стоп.

Следующие три функции обслуживания предназначены для манипулирования символами. С помощью этих функций осуществляются операции ввода-вывода без непосредственного обращения из пользовательских программ к области памяти,в которой хранится закодированный образ экрана дисплея. В связи с уже упоминавшимся ранее аспектами вывода информации на экран дисплея путем прямой модификации содержимого памяти, хранящей копию изображения, здесь следует заметить, что испоьзование этих функций делает программы пользователя более мобильными и менее зависимыми от особенностей Вашего компьютера. Рассмотрим эти функции более подробно.

Девятая функция обслуживания (функция обслуживания с кодом 8) осуществляет считывание символа и атрибута, соответствующих текущей позиции курсора. Как и в случае некоторых других функций обслуживания, здесь следует указать номер используемой страницы (несмотря на то, что страница применяется только к текстовому режиму цветного графического дисплея). Функция обслуживания 8 работает как в графическом, так и в текущем режимах; более детальные сведения, касающиеся отображения символов в графическом режиме приведены в следующей главе. Заметим лишь, что поскольку в графическом режиме атрибут изображения отсутствует (это понятие применимо лишь к текстовому режиму, либо к монохромному дисплею), то его считывание не осуществляется.

Особенность этого режима состоит в том, что одни и те же таблицы изображения символов, используемые для их записи, используются также и в случае их распознавания путем сличения. Распознование осуществимо даже в случае создания своего собственного набора графических символов.

Код символа ASCII заносится в регистр AL. Если символ не соответствует кодам ASCII, то в графическом режиме в регистр

заносится 00. Атрибуты изображения заносятся в регистр AH. Страница изображения указывается в регистре BH. Для рабюоты в графическом режиме установку страницы осуществлять не следует.

Номер функции обслуживания	Параметры
AH = 8	BH = активная страница изображения (для графического режима не требуется) AL = счимвол ASCII, считанный из позиции, указанной курсором AH = атрибут текстового символа

Функция обслуживания с кодом 9 предназначена для записи символа (или нескольких его копий) и атрибута. Символ задается в регистре AL, а атрибут текстового режима или цвет графического режима заносится в регист BL. Количество копий символов помещается в регистр CX.

Для текстовых режимов в регистре BH должна быть указана страница изображения; графические режимы этого не требуют.

Символ и его атрибуты цвета записываются столько раз, сколько требуется (начиная с текущей позиции курсора). Курсор при этом не перемещается, запись производлится в последующие позиции экрана. В текстовом режиме при дублировании символов может быть организован переход на следующую строку. В графическом режиме такая возможность отсутствует.

Описываемая функция обслуживания весьма полезна как в одиночном так и групповом режимах. Групповой режим часто используется для оперативного вывода пробелов или других повторящихся символов (например, горизонтальных линий). Если требуется сформировать символ в единственном экземпляре, то следует установить счетчик в CX равным 1. Если его значение равно 0, то число повторений есть величина переменная.

Отличия функции обслуживания с кодом 9 от функции обслуживания с кодом 14 состоит в следующем: имеется возможность управлять атрибутами цвета, отсутствует автоматическое отображение курсора при записи символа.

В графическом режиме цвет указанный в регистре BL,

является цветом собственно пикселя, входящего в состав рисунка символа. Если седьмой бит равен 1, то биты в BL, определяющие цвет, объединяются с битами цвета текущего пикселя при помощи операции "исключающего или" (XOR). Такой подход гарантирует отличие полученного цвета от прежнего, обеспечивая четкость изображения. Если 7-й бит в регистре BL равен 0, то цвет в BL заменяет существующие цвета пикселей. Сказанное справедливо также в отношении функций обслуживания с кодами 10 и 12.

Номер функции обслуживания	Параметры
AH = 9	AL = символ ASCII, предназначенный для вывода на экран BL = атрибуты символа, выводимого на экран BH = активная страница изображения (в графических режимах отсутствует) CX = число записываемых символов и атрибутов

Функция обслуживания с кодом 10 ("запись символа") отличается от предыдущей только тем, что не позволяет в текстовом режиме изменять существующие атрибуты цвета. Однако для графического режима регистр BL должен содержать атрибуты цвета. Таким образом название этой функции не совсем корректно. На функции обслуживания с кодами 9 и 12 распространяются одни и те же правила работы с цветом в графических режимах: любой цвет можно использовать либо непосредственно, либо в комбинации с существующим цветом (операция XOR).

Номер функции обслуживания	Параметры

AH = 10 AL = символ ASCII, выводимый на экран
BL = атрибуты цвета для графических режимов
BH = активная страница
CX = число записываемы символов

Функция обслуживания с кодом 11 ("установка палитры") предназначена для выбора одной из двух графических палитр среднего разрешения. Прежде чем передать управление этой функции обслуживания следует загрузить в регистр BH идентификатор палитры, а в регистр BL - значение цвета.

Напомним здесь, что в графических режимах (режимы 4 - 6) каждый пиксель на экране имеет свой цвет. Цвет устанавливается в основном тем же способом, что и атрибуты в текстовых режимах, однако имеются некоторые отличия. Во-первых, пиксели не могут мерцать. Во-вторых, поскольку каждый пиксель является дискретной цветной точкой, то не существует ни очертания, ни фона; посто каждый пиксель имеет тот или иной цвет. Когда, в графическом режиме формируется текст, то один цвет используется для пикселей, из которых составляется фон, а другой для пикселей, из которых состовляются символы.

Примечание: Специфика использования графического режима в языке Бейсик дает основание считать, что здесь возможен фоновый цвет. На самом деле это не так.

Для каждого графического режима существуют заранее определенные наборы цветов, известные как палитры. Стандартные палитры можно изменять в компьютере IBM PC/jr или в установленом графическом адаптере; обычный цветной графический адаптер такой возможности не имеет. Если цвета палитры для любого графического адаптера определены, то цвет каждого пикселя выбирается из имеющихся цветов путем установки значений битов, определяющих цвет пикселя. В 2-х цветном режиме имеется один бит для каждого пикселя и цвет пикселя задается в виде О или 1. В 4-х цветном режиме предусмотрены два бита, принимающие значения от О до 3. В 16-ти цветном режиме цвет задается с помощью 4 битов, хранящих значение от 0 до 15.

Одна из модификаций функции обслуживания с кодом 11 применима к текстовым режимам, все остальные - только к графическим. Для текстовых режимов справедливо следующее правило: если BH=0, то BL указывает цвет окружения текстовой

области, выбранной из полной 16-ти цветной палитры. Для любого графического режима, если BH=0, то BL указывает цвет фона и цвет окружения. Окружение эквивалентно произвольной области экрана, если речь идет о цвете фона. Значение BL может быть выбрано из полной 16-ти цветной палитры.

Если BH=1, то BL определяет палитру. Для цветного графического адаптера это применимо только к режиму 4 (среднее разрешение 4-х цветная графика). Для более совершенных графических адаптеров, в том числе и для PCjr, это положение расространяется и на все остальные режимы. Здесь мы рассматриваем только стандартные 4-х цветные палитры, которые характерны для режима 4 (палитры 0 и 1).

Палитра 0:

0 - текущий цвет фона
1 - зеленый (2)
2 - красный (4)
3 - коричневый (6)

Палитра 1:

0 - текущий цвет фона
1 - циан (3)
2 - пурпупный (5)
3 - белый (7)

Номер функции обслуживания	Параметры
AH = 11	BH = указатель палитры (0 или 1 для среднего разрешения BL = цвет или палитра использования с указателем цвета

Функция обслуживания с кодом 12 (C/16) предназначена для записи пикселя. Поскольку положение курсора, используемое функциями обслуживания с кодами 9,10,14, рассматривается в контексте символов, то здесь необходимо указывать строку

развертки и позицию (столбец) пикселя. Координаты пикселя отсчитываются от верхнего левого угла экрана (0,0). Номер строки развертки, для которого треуется один байт указывается в регистре DL, для номера столбца одного байта недостаочно - ему отведен регистр CX.

Цвет задается в регистре AL и может быть использован либо непосредственно, либо в сочетании с другими цветами (операция XOR).

Номер функции обслуживания	Параметры
AH = 12	AL = код цвета пикселя (0 - 15) DL = номер строки пикселя CX = номер столбца пикселя

Функция обслуживания с кодом 13 (D/16) предназначена для чтения пикселя, точнее для получения информации о его цвете. (Для сравнения напомним, что процедура обслуживания с кодом 8 формирует на выходе как код цвета, так и код символа ASCII). Код цвета пикселя формируется в регистре AL; старшие биты при этом устанавливаются в 0. Регистры DL и CX предназначены для передачи процедуре обслуживания номера строки и номера столбца пикселя.

Номер функции обслуживания	Параметры
AH = 13	AL = код цвета пикселя (0 - 15) DL = номер строки пикселя CX = номер столбца пикселя

Функция обслуживания с кодом 14(E/16) предназначена для записи символа в режиме телетайпа. При этом экран дисплея оказывается в роли принтера или пишущей машинки. Такие команды операционной системы как TYPE или COPY (если в качестве результирующего файла указана консоль "CON") используют эту функцию обслуживания для имимтации принтера или пишущей машинки. В сущности, в рамках этой функции обслуживания возможности аппаратуры в части генерации цвета, мерцания или управления курсором не используются.

При обращении к процедуре обслуживания с кодом 14 производится запись символа в текущую позицию (указываемую курсором) и осуществляется перемещение курсора. При необходимости выполняется переход к новой строке или сдвиг содержимого экрана. Записываемый символ указывается в регистре AL. Для текстового режима текущие атрибуты экрана сохраняются. Для графического режима цвет фона указывается в регистре BL.

Из всего набора символов следующие четыре символа непосредственно не отображаются, а выступают в роли упрвляющих (интепретируются аппаратурой): CHR$(7) - звук, CHRS(8) - возврат на одну позицию, CHR$(10) - перевод строки, CHR$(13) - возврат каретки.

Номер функции обслуживания	Параметры
AH = 14	AL = записываемый символ ASCII BL = цвет символа (только для графических режимов) BH = страница (для графических режимов не указывается)

Приложение 8.1. Текст программы демонстрации возможностей управления цветом (Бейсик).

```
1000 REM Listing 8.1
1010 REM
1020 REM
1030 GOSUB 2000 ' TITLE
1040 GOSUB 3000 ' GET WHICH DISPLAY TYPE, AND SET ADDRESS
1050 GOSUB 2000 ' SET THE TITLE AGAIN
1060 GOSUB 4000 ' BUILD THE SURROUNDING COMMENTS
1070 GOSUB 5000 ' BUILD THE DISPLAY ARRAY
1080 GOSUB 6000 ' FINISH UP AND RETURN TO DOS

2000 REM Title subroutine
2010 KEY OFF : CLS : WIDTH 80
2020 REM
2030 PRINT "         Programs for INSIDE THE IBM PERSONAL COMPUTER"
2040 PRINT "         (C) Copyright 1983 Peter Norton"
2050 PRINT
2060 PRINT     Program 8-1: Demonstrate all screen attributes"
2999 RETURN

3000 REM Subroutine to inquire about display type
3010 PRINT
3020 PRINT "Before we go any further, is this a color-graphics display? ";
3030 GOTO 3060
3040 PRINT
3050 PRINT " (answer Y or N) ";
3060 ANSWER$=INKEY$
3070 IF LEN(ANSWER$)<1 THEN 3060
3080 IF LEN(ANSWER$)>1 THEN 3040
3090 SEGVAL!=0
3100 IF MID$(ANSWER$,1,1)="Y" THEN SEGVAL!=&HB800 ' Color segment
3110 IF MID$(ANSWER$,1,1)="y" THEN SEGVAL!=&HB800 ' Color segment
3120 IF MID$(ANSWER$,1,1)="N" THEN SEGVAL!=&HB000 ' Monochrome segment
3130 IF MID$(ANSWER$,1,1)="n" THEN SEGVAL!=&HB000 ' Monochrome segment
3140 IF SEGVAL!=0 THEN 3040
3150 DEF SEG=SEGVAL!
3999 RETURN

4000 REM  subroutine to build the surrounding messages
```

```
4010 LOCATE 11,1  : PRINT "Normal"
4020 LOCATE 12,1  : PRINT "normal"
4030 LOCATE 19,1  : PRINT "Normal"
4040 LOCATE 20,1  : PRINT "blinking"
4050 LOCATE 11,69 : PRINT "Bright"
4060 LOCATE 12,69 : PRINT "normal"
4070 LOCATE 19,69 : PRINT "Bright"
4080 LOCATE 20,69 : PRINT "blinking"
4090 FOR HEX.DIGIT%=0 TO 15
4100   LOCATE 6,HEX.DIGIT%*3+17
4110   PRINT HEX$(HEX.DIGIT%)
4120   LOCATE HEX.DIGIT%+8,11
4130   PRINT HEX$(HEX.DIGIT%)
4140 NEXT HEX.DIGIT%
4150 LOCATE ,,0
4999 RETURN

5000 REM subroutine to set the display array
5010 FOR ROW%=0 TO 15
5020   FOR COL%=0 TO 15
5030     POKE(ROW%+7)*160+COL%*6+31,ROW%*16+COL%
5040     POKE(ROW%+7)*160+COL%*6+33,ROW%*16+COL%
5050     POKE(ROW%+7)*160+COL%*6+35,ROW%*16+COL%
5060     POKE(ROW%+7)*160+COL%*6+32,ROW%*16+COL%
5070   NEXT COL%
5080 NEXT ROW%
5999 RETURN

6000 LOCATE 25,1,1
6010 PRINT "Press any key to return to DOS... ";
6020 IF LEN(INKEY$)=0 THEN 6020 ' wait to keystroke
6030 CLS
6999 SYSTEM

9999 REM End of program Liisting 8-1
```

Приложение 8.2. Текст программы генерации изображений (Паскаль).

```
module Listing_8_2;
```

```
{=====================================================
====}

type

  screen_position_pair_type=(character_byte,attribute_byte);
  normal_screen_type=array[1..25,1..80,
        character_byte..attribute_byte] of char;

  narrow_screen_type=array[1..25,1..40,
        character_byte..attribute_byte] of char;

{=====================================================
====}

var [static]

  screen_pointer     : ads of normal_screen_type;

  current_attribute  : char;
  row,column         : integer;
  output_string      : lstring(255);

  page               : word;
  eighty_col_mode    : boolean;

{=====================================================
====}

const
  normal_attrib = chr   (7);
  reverse_video = chr (112);
  bright        = chr  (15);
  blinking      = chr (135);
  alert         = chr (140);

{=====================================================
====}

procedure clear_reset;
  external;
function video_mode : byte;
  external;

{=====================================================
====}
```

```
procedure set_video_address;
  begin
    if video_mode=7 then
      begin
        screen_pointer.s:=#B000;
        screen_pointer.r:=0;
      end
    else
      begin
        screen_pointer.s:=#B800;
        screen_pointer.r:=0;
      end;
  end;

{=================================================
====}

procedure clear_screen;
  begin
  {if you have the assembly language routines, just to this: }
    clear_reset;
    return;
  {otherwise, we'll clear the screen the hard way: }
    for row:=1 to 25 do
      for column:=1 to 80 vdo
        begin
          screen_pointer^[row,column,character_byte]:=' ';
          screen_pointer^[row,column,attribute_byte]:=normal_attrib;
        end;
  end;

{=================================================
====}

{                НЕРАЗБОРЧИВО                    }

{=================================================
====}

procedure set_pointer_for_page;
  begin
    if eight_col_mode then
      screen_pointer.s:=#B800+page*4096
    else
      screen_pointer.s:=#B800+page*2048
```

```
  end;

end.
```

ГЛАВА 9. ВИДЕОДОСТУП 2 - ГРАФИЧЕСКИЙ РЕЖИМ

[Оглавление]

После завершения обсуждения основ видеопредставления информации и собственно текстового режима в предшествующей главе, перейдем к рассмотрению особенностей работы в графическом режиме.

Не следует удивляться сложности предмета "машинная графика". Сама по себе графика - вещь непростая; здесь же она осложняется чисто техническими аспектами, такими как цветовые палитры, многочисленные режимы м хранимые образы содержимого
экрана. Это сложный, многоаспектный вопрос. В рамках данной главы мы не будем углубляться в детали. Если в Ваши намерения входит использование графического режима IBM/PC или просто изучение основ предмета, то следует продолжать чтение. В противном случае, материал настоящей главы может быть опущен.

Все сказанное ниже распространяется на компьютер IBM/PC, а также на все совместимые с ним компьютеры (последние предполагают возможность выполнения произвольной программы, в том числе и графической, написанной для IBM/PC). Если речь идет о компьютерах, использующих операционную систему MS-DOS,
то большинство положений нуждается в уточнениях.

9.1. Основы машинной графики

Существует два основных способа формирования графических изображений на видеомониторе. Причудливый мир компьютерных (экранных) игр позволяет получить о них какое-то представление, оценить их место, роль и возможности. (Ниже об этом еще пойдет речь, однако следует все это увидеть собственными глазами.)

Эти способы основываются либо на векторном, либо на растровом (точечном, пиксельном) принципах формирования изображений, далее именуемых просто векторной или пиксельной графикой. Графика IBM/PC - это пиксельная графика; именно этому виду графики мы уделим основное внимание.

(Термин "вектор", кстати, уже использовался нами при рассмотрении прерываний. Там он обозначал два машинных слова,

содержащих адрес программы, выполняющей обработку прерываний;
здесь под "вектором" понимается отрезок прямой. И первая, и вторая интерпретация термина имеют чисто математические параллели, с которыми мы не будем далее соприкасаться. Здесь нам важно эти термины не смешивать).

При построении изображений на экране дисплея средствами векторной графики используются только прямые. В этом случае машинная программа специфицирует две крайние точки прямой, а собственно дисплей "проводит" между ними прямую.

Если взять обычный экран дисплея - будь то телевизионный приемник или компьютерный терминал - то в случае растрового сканирования электронный луч, генерирующий изображение перемещается по стандартному шаблону, образуемому строками развертки, заполняющими весь экран. В случае дисплея, работающего на принципах векторной графики, перемещение электронного луча осуществляется по команде, рисующей заданную прямую между заданными точками. В рамках системы векторного сканирования, перемещение электронного луча осуществляется по заданной в каждом случае траектории, а не по раз и навсегда установленному шаблону.

Векторная графика обладает рядом существенных преимуществ - ей присущи высокая четкость, и точность, а также сравнительно высокое быстродействие. Все это делает ее эффективной в сфере инженерной графики, а также при построении особенно сложных или точных графических дисплеев. Однако векторно-графические системы используются лишь для вычерчивания прямых (или изображений, которые могут быть представлены в виде совокупности коротких отрезков).В рамках этой системы оказывается невозможным заполнять изображения цветом сложной формы. Это сильно сужает область их возможных применений.

В основе растровой или пиксельной графики лежит другой принцип. Экран дисплея здесь разделяется на прямоугольную сетку, состоящую из множества мельчайших элементов изображения, называемых также пикселями или пэлами. Каждый пиксель обладает свойством светимости. Таким образом изображение синтезируется из множества отдельных точек.

Размеры пикселей и расстояния между ними тщательно рассчитываются таким образом, чтобы промежутки между ними отсутствовали. Если группа смежных пикселей находится в возбужденном состоянии (т.е. светится), то они воспринимаются глазом не как совокупность отдельных точек, а как сплошной участок. Если бы точки были достаточно малы, то этот тип дисплея мог бы конкурировать с векторным по части четкости и точности, однако, у большинства растровых дисплеев этот

показатель оставляет желать лучшего.

В принципе, любая система машинной графики может обеспечить воспроизведение цвета, однако, чаще всего, цвет ассоциируется с растровой графикой. Цвет - это веский (но не единственный) аргумент в пользу превосходства систем растровой графики над системами векторной графики в игровой и деловой сферах. Способность воспроизводить сплошные, прямо-криволинейные поверхности в цвете делает растровую графику намного предпочтительней векторной (исключение составляют специальные приложения, например, технические чертежи).

И последний аспект. В отличие от систем векторной графики, в которых изображение генерируется с помощью последовательности команд (каждая команда рисует одну небольшую прямую), системы растровой графики допускают использование хранимого в памяти образа экрана (эти вопросы мы уже рассматривали применительно к текстовым образам). В компьютере IBM/PC мы имеем дело именно с системой растровой (пиксельной) графики; управление процессом построения изображения на экране дисплея осуществляется с помощью хранимого образа экрана. Все преимущества дисплеев с хранимыми образами экрана распространяются на растровый дисплей компьютера IBM/PC - и в первую очередь способность к считыванию и записи отображаемых данных непосредственно из памяти дисплея.

9.2. Понятие элемента отображения (пикселя)

По мере движения сканирующего луча по экрану дисплея высвечиваются те или иные пиксели. (По крайней мере те пиксели, которые находятся в видимой части экрана; дело в том, что сканирующий луч выходит за пределы видимой части экрана, в так называемую "закадровую" область. Эта область не содержит пиксели, однако, для нее может быть установлен цвет. В языке Бейсик для этого служит параметр границ оператора COLOR (ЦВЕТ); кроме того, одна из функций обслуживания BIOS-ПЗУ может быть использована для установки закадрового цвета).

Напомним, что в текстовом режиме экран персонального компьютера IBM/PC состоит из 25 горизонтальных текстовых строк; из материалов предшествующей главы, связанных с описанием курсора, известно также, что на каждый символ цветного графического дисплея приходится восемь горизонтальных строк развертки. Отсюда следует, что вертикальный размер общего пиксельного поля IBM/PC равен 200 строкам (8X25).

Горизонтальный размер пиксельного поля может

варьироваться. Этот размер зависит от объема памяти, требуемой для управления пикселями, а также от величины пикселя (точки), высвечиваемой на том или ином графическом мониторе.

Компьютер IBM/PC имеет два различных горизонтальных формата (размера) пиксельного поля, устанавливаемых в зависимости от требуемого размера (а, следовательно, и количества) пикселя(ей). В режиме среднего разрешения ширина экрана составляет 320 точек, а в режиме высокого разрешения - 640 точек. (Компьютер IBM/PC допускает также возможность работы в режиме низкого разрешения, однако, этот режим практически не используется. В режиме низкого разрешения поверхность экрана представляется в виде матрицы 160X100 пикселей. Аппаратура цветного графического адаптера поддерживает режим низкого разрешения, однако, программная поддержка на уровне BIOS-ПЗУ отсутствует).

Таким образом, графический режим среднего разрешения предусматривает наличие 320 точек по горизонтали экрана и 200 точек по вертикали, в то время как графический режим высокого разрешения - 640 точек по горизонтали и 200 точек по вертикали. Понятно, что изображение, представленное в режиме высокого разрешения, способно воспроизводить более мелкие детали только на горизонтали. Для большинства приложений качество изображения практически осталось бы неизменным, если бы режим высокого разрешения выражался параметрами 320X400.

Каждый пиксель на экране дисплея может находиться либо в активном (светимость), либо в пассивном состоянии; в режиме среднего разрешения может быть воспроизведен и цвет. В режиме высокого разрешения цвет не воспроизводится, отчасти из-за особенностей генерации изображения, отчасти из-за отсутствия дополнительной памяти, необходимой для воспроизведения цвета.

Рассмотрим требования к памяти, предъявляемые графическим режимом высокого разрешения. Если к числу сведений, сообщаемых о пикселе, относится только признак активного или пассивного состояния (наличие или отсутствие светимости пикселя) - а именно это и сообщается о пикселе -, то для управления пикселем достаточно одного бита, принимающего значение 0 или 1. В режиме высокого разрешения имеется 640X200 (или 128000) пикселей. Для управления ими требуется 128000 бит, а с учетом того, что байт состоит из восьми битов, - 16000 байт. Именно такая память выделена цветному графическому адаптеру (округлена до 16К).

Режим среднего разрешения требует вдвое меньшего числа пикселей. Имея те же 16К байт памяти мы можем выделить для каждого пикселя два бита. С помощью двух битов могут быть представлены четыре различные величины (0, 1, 2, 3); но это

позволит лишь частично решить проблему воспроизведения цвета.

Прежде, чем познакомиться с окончательным разрешением проблемы воспроизведения цвета в графическом режиме, обратим внимание на одно отличие в специфике использования памяти дисплея в графическом и текстовом режимах. Из предыдущей главы нам известно о наличии двух текстовых режимов - 80-ти позиционного и 40-позиционного. Графика представлена также двумя режимами: режимом среднего разрешения и режимом высокого разрешения. В обоих случаях число отображаемых элементов одного режима вдвое превышает число отображаемых элементов другого режима. Соответственно изменяются и требования, предъявляемые к памяти. На этом сходство между ними заканчивается. Если в текстовом режиме дополнительная память используется для увеличения числа страниц, то в графическом режиме дополнительная память используется для удвоения объема памяти, отводимой под один пиксель.

В связи со спецификой управления цветом в компьютере IBM/PC возникает вопрос о количестве одновременно используемых цветов. Цветной графический адаптер и экран видеомонитора способны воспроизвести любой из шестнадцати возможных цветов для каждого пикселя. Здесь нет никаких проблем. Проблемы возникают при спецификации цвета.

Фирма IBM могла пойти по пути создания цветного графического адаптера, имеющего достаточно памяти для представления всех возможных цветов каждого пикселя, однако она этого не сделала. Как уже говорилось выше, в режиме высокого разрешения каждому пикселю отводится только один бит, поэтому в принципе в этом режиме невозможно воспроизвести более двух цветов - белого и черного. В режиме среднего разрешения каждому пикселю отводится два бита, поэтому возможно воспроизведение четырех цветов.

Проблема выбора того или иного набора, состоящего из четырех цветов, решена необычно и остроумно. В любой момент времени активны только четыре цвета, но пользователю предоставляется возможность установить любой из четырех цветов по своему усмотрению. Остальные три устанавливаются в рамках так называемой палитры. В качестве компенсации фирма IBM предоставляет на выбор две возможные палитры - первая состоит из зеленого, красного и коричневого (на большинстве дисплеев воспроизводится как желтый), а вторая - из циана, малинового и белого.

Если все это звучит недостаточно убедительно, давайте рассмотрим этот вопрос с другой стороны. При задании цвета любого пикселя в режиме среднего разрешения мы используем один из четырех кодов 0, 1, 2, 3. Что стоит за этими числами? Ничего жестко фиксированного. Их смысл определяется в

процессе спецификации используемого цвета. Мы можем закрепить за кодом 0 один из 16 цветов. Для кодов 1-3 фирма IBM предлагает две стандартные возможности (две палитры), и пользователь должен выбрать, какую палитру он будет использовать. Заметим здесь, что пользователь не может влиять на составление палитры.

Давайте еще раз осмыслим происходящее. Если бы объем памяти был достаточно велик, то мы могли бы для каждого пикселя использовать полную гамму цветов (16). В действительности объем памяти таков, что в режиме высокого разрешения цветовая гамма каждого пикселя состоит только из двух цветов, а в режиме среднего разрешения - только из четырех. Пользователю могли бы быть предоставлены более широкие возможности в части выбора цвета, однако по этому пути разработчики персонального компьютера не пошли.

В режиме высокого разрешения пользователь лишен возможности выбирать конкретные цвета (два цвета) по своему усмотрению; этими цветами могут быть только черный и белый. В режиме среднего разрешения пользователь имеет некоторую свободу выбора цветовой гаммы. Из четырех цветов один цвет устанавливается пользователем произвольно. Остальные три цвета устанавливаются в рамках одной из двух строго фиксированных палитр.

Таким образом, любой цветной дисплей в графическом режиме позволяет использовать четыре цвета, причем три цвета выбираются из палитры 1 (зеленый, красный, коричневый), либо из палитры 2 (циан, малиновый, белый).

Если цветной дисплей находится в графическом режиме, то цвет любого пикселя (из состава имеющихся четырех цветов) можно применить путем изменения содержимого двух битов, описывающих этот пиксель. Существует и возможность изменить сразу цвета всего изображения; для этого следует: либо изменить палитру, либо изменить цвет 0, либо изменить и то и другое. В этих случаях собственно изображение (картинка) не меняется, изменяется лишь ее цвет(а). (Для графического режима низкого разрешения предусмотрено достаточно битов, чтобы каждый пиксель мог воспроизвести любой из 16 возможных цветов. Режим низкого разрешения может быть вполне реализован на Вашем компьютере; аппаратная поддержка этого режима реализована полностью. Однако программная поддержка на уровне BIOS-ПЗУ полностью отсутствует. Если функции программ BIOS возьмет на себя программа пользователя, то графический режим низкого разрешения будет функционировать).

9.3. Отображение пикселей на экране

Графический режим работы компьютера представляет собой наиболее сложный режим. Если использовать процедуры BIOS-ПЗУ, ориентированные на обслуживание графического режима (описаны в параграфе 8.9), то декодирование образа экрана выполняется автоматически. Это одна из причин их широкого использования. В тех случаях, когда необходимо использовать режим прямого управления, следует детально ознакомиться с материалом настоящего параграфа.

Подобно обычному телевизору, видеомонитор генерирует изображение за два прохода. На первом проходе электронный луч перемещается только по четным строкам развертки; после этого (на обратном ходе луча) сканируются все нечетные строки.

При работе в текстовом режиме это обстоятельство в расчет не принимается, поскольку функции управления формированием изображения полностью выполняет контроллер видеомонитора. В графическом режиме эти особенности необходимо учитывать.

Электронные схемы генерации изображения в графическом режиме должны воспринимать поток битов, определяющих пиксели, причем воспринимать их в том порядке, в котором сканирующий луч перемещается по экрану - сначала по четным, а затем по нечетным строкам. По этой причине образ экрана хранится не в том виде, который удобен человеку-оператору или программе; структура хранимого образа отвечает потребностям электронных схем генерации. На рис.9.1 воспроизводится эта структура.

Для того, чтобы структура хранимого образа экрана соответствовала механизму сканирования экрана, память дисплея организована в виде двух блоков, смещенными друг относительно друга на 4К. Первый блок содержит информацию о пикселях четных строк,а второй блок - о пикселях нечетных строк. Рассмотрим эти вопросы более подробно (используя в качестве примера графику с высоким разрешением).

В начале области (блока) памяти располагаются биты, управляющие процессом отображения на экране дисплея первой строки, т.е. строки с номером 0. Строка содержит 640 пикселей (графический режим с высоким разрешением), а каждый пиксель требует для индикации своего состояния 1 бит. Таким образом, для управления первой строкой требуется 640 битов или 80 байтов. Для управления следующей четной строкой (строкой с номером 2) также требуется 640 битов или 80 байтов и т.д. Самый первый бит управляет точкой экрана дисплея, находящейся на пересечении нулевой строки и нулевого столбца (0,0), следующий бит управляет точкой (0,1) и т.д. Последней точкой первой строки является точка (0,639); следом за ней идет

точка (2,0) и т.д. В конце концов блок памяти, хранящий четные строки, оказывается исчерпанным, и с нового адреса, кратного 1К, начинается блок памяти, хранящий нечетные строки.

Для графического режима со средним разрешением память организована точно также, с той лишь разницей, что горизонтальная координата экрана состоит из 320 пикселей, причем каждый пиксель представлен двумя битами, позволяющими представить цветовые комбинации. Для представления строки по-прежнему требуется 80 байтов; для представления используются два блока памяти: один - для четных, а другой - для нечетных строк.

Заметим здесь, что конечный адрес памяти будет одним и тем же как для режима среднего, так и для режима высокого разрешения. Но в одном случае число пикселей вдвое меньше (при вдвое большем количестве битов, отведенных для

представления каждого пикселя), а в другом случае вдвое больше (соответственно, количество битов, отведенных для представления пикселя в два раза меньше). Такой подход позволяет сохранить постоянную длину строки для обоих режимов - 640 битов или 80 байтов. Программы, приведенные в приложении (листинг 9.1), дают возможность пользователям, программирующим на Паскале, эффективно управлять любым пикселем. Эти программы осуществляют необходимые арифметические преобразования, связанные с пересчетом позиции пикселя на экране дисплея в адрес памяти, хранящей кодированный образ экрана, избавляя пользователя от рутинной работы по вычислению смещений битов и байтов.

Эффективность этих процедур становится особенно очевидной, когда требуется создавать быстродействующие программы генерации изображений. Любая программа, относящаяся к указанному классу, должна тем или иным способом переводить пиксели из активного состояния (состояние светимости) в пассивное и наоборот. Сделать это можно либо используя функции обслуживания BIOS-ПЗУ, описанные в предыдущей и в последней главах, либо упомянутые Паскаль-процедуры (листинг 9.1). Высокое быстродействие программ генерации изображений предполагает, что тот или иной фрагмент будет загружен в память графического адаптера сразу целиком, а не будет формироваться побитно. А для этого необходимо произвести декомпозицию изображения на четные и нечетные строки, после чего обеспечить их размещение в соответствующих блоках памяти. Указанное обстоятельство сильно усложняет процесс быстрой генерации изображений.

Решение задачи быстрой генерации изображений зависит от

потребностей и от принятых компромиссов. Один из возможных подходов состоит в разбиении изображения на участки, помещаемые в один байт (четыре пикселя для режима среднего разрешения и восемь пикселей для режима высокого разрешения). Приняв такую схему, можно осуществлять загрузку фрагментов в память графического адаптера побайтно. Такой метод позволяет резко снизить вычислительные затраты, в противном случае, программы должны выполнять массу операций манипулирования битами, таких, например, как сдвиги, логические "И", логические "ИЛИ" и т.п. Работа с байтами повышает скорость генерации образа приблизительно в три раза, при условии, что собственно образ уже построен.

Существует и другой, более грубый способ повышения скорости генерации изображения. Его суть состоит в использовании текстовой графики (псевдографики), описываемой в следующем параграфе.

9.4. Представление текста (символов) в графическом режиме.

Отображение символов на экране дисплея не представляет особого труда, даже если компьютер работает в графическом режиме. Символы в этом случае представляются в виде совокупности пикселей, как если бы это были графические объекты, подлежащие рисованию (вычерчиванию).

Для фиксации схемы вычерчивания символов в графическом режиме приняты специальные соглашения. Площадь, занимаемая символом, представляется в виде матрицы размером 8X8 пикселей. Принятый принцип кодирования в принципе очевиден. Для воспроизведения очертания (рисунка) символа отводится восемь байтов. Восемь битов каждого байта управляют состоянием восьми пикселей одной строки рисунка символа (нумерация пикселей ведется слева направо). Очередной байт описывает очередную строку (нумерация строк осуществляется сверху вниз).

На рис.9.2 представлена диаграмма отображения символов (букв), содержащая сведения об особенностях взаимодействия с таблицей рисования символов. В качестве примера выбрана буква "А".

Даже при беглом знакомстве с этой диаграммой возникает ряд интересных вопросов. Нижняя строка диаграммы отведена под нижний выносной элемент символа (литеры), такой, например, как "хвост" прописной буквы "у". Две верхние строки отведены под заголовок литеры. В качестве литер, имеющих заголовок, можно привести строчную букву "ь" (имеется в виду

ее вертикальный отрезок, использующий область заголовка), а также все прописные буквы алфавита (буквы верхнего регистра). Вопреки "здравому смыслу" буквы размещаются не по центру матрицы, а смещаются (выравниваются) по ее левой границе, так что два столбца справа служат для отделения символов друг от друга.

При формировании символов в графическом режиме BIOS-ПЗУ осуществляет поиск рисунка (образа) символа во внутренней таблице отображения, а затем найденное табличное описание использует для команд активизации пикселей. На заключительной стадии устанавливаются соответствующие биты в памяти дисплея путем пересылки четных и нечетных строк в соответствующие блоки памяти.

Поскольку в графическом режиме высокого разрешения цвет не воспроизводится (воспроизводятся только черный и белый цвета), то рисунок (образ) символа непосредственно управляет активизацией пикселя. В цветном режиме среднего разрешения воспроизводятся цвет очертания (рисунок) и фоновый цвет. Фоновый цвет - это цвет, обозначенный кодом 0; этим кодом может быть обозначен любой из 16-ти имеющихся цветов. Цвет очертания устанавливается с помощью кода 3 текущей цветовой палитры; для палитры 1 это белый цвет, а для палитры 0 - коричневый/желтый.

Процедуры манипулирования символами BIOS-ПЗУ могут осуществлять как считывание, так и запись символов. Функция обслуживания с кодом 8 является именно такой процедурой. В текстовом режиме никаких проблем при чтении не возникает, поскольку в памяти дисплея хранится байт с кодом ASCII требуемого символа. В графическом режиме все обстоит не так просто, поскольку байт с кодом ASCII в памяти отсутствует. Вместо этого память содержит группу байтов, представляющих точечный (пиксельный) рисунок символа. Поэтому процедура BIOS-ПЗУ чтения символа осуществляет нетривиальные операции. Она читает пиксели и сопоставляет их с содержимым таблицы рисунков (образов) символов, выполняя своего рода распознавание образа.

Таблица рисунков (образов) символов состоит из двух частей и представляет для пользователя значительный интерес. Первая часть этой таблицы хранится в ПЗУ и содержит первые 128 символов кода ASCII, то есть все стандартные символы алфавита. Вторая часть таблицы содержит рисунки остальных 128 символов (т.е. символов CHR$(120)-CHR$(255)). Поскольку первая часть находится в ПЗУ (указатель на нее строго фиксирован), то она не может быть ни модифицирована, ни заменена на другую таблицу. Вторая часть таблицы, в отличие от первой, ориентирована на пользователя; она создается

пользователем под его потребность.

Один из векторов прерывания (прерывание с кодом 31 (1F(16)), размещенный в таблице векторов по адресу 124(7C(16)) используется для указания на эту таблицу. В исходном состоянии таблица отсутствует и вектор равен нулю. После создания нужной таблицы и загрузки ее адреса в вектор прерывания любое обращение к BIOS-ПЗУ для записи символа с кодом CHR$(128)-CHR$(255) приведет к воспроизведению на экране этого символа.

Такой подход, основанный на свободном определении таблицы рисунков 128 символов является важной предпосылкой для быстрого построения наиболее типичных графических образов (несложные графики, схемы, рисунки, таблицы, чертежи, диаграммы и др.). Если требования к изображаемым объектам не слишком высоки, то всегда можно определить ансамбль, состоящий из 128 шаблонов (каждый шаблон имеет размер 8X8 пикселей) и генерировать на их основе эти объекты. Очевидно, что наиболее сложные и детальные образы не могут быть построены таким способом. И все-таки ансамбль, состоящий из 128 предварительно определенных шаблонов, достаточно велик, чтобы пользовательские программы обладали высоким быстродействием, были просты и компактны. В сущности, процедуры обслуживания рисунков символов BIOS-ПЗУ (используя таблицу, созданную пользователем) берут на себя основную работу по генерации образов; пользовательские программы лишь помещают нужные символы в нужные позиции экрана.

Описанные возможности генерации графических символов отнюдь не исключают возможность непосредственного доступа к памяти дисплея и управления отдельным пикселем. В графическом режиме пользователь может сочетать обе возможности. Когда это удобно - использовать таблицу графических символов, при необходимости - манипулировать пикселями. Вполне допустимо, например, изменять любой пиксель внутри любого образа символа, уже выведенного на экран.

Таблица рисунков символов требует для своего размещения 1024 байтов памяти: 128 символов X 8 байтов (требуемых для каждого символа). В графическом режиме со средним разрешением один байт программы пользователя может использоваться для управления образом, состоящим из 64 пикселей, что эквивалентно восьми байтам отображаемой информации. При соблюдении определенных требований такой подход позволяет в восемь раз сжать графический образ по отношению к исходному.

Прежде чем использовать эти возможности, следует оценить последствия ограничений, распространяемых не только на число возможных шаблонов, но и на число воспроизводимых цветов в режиме среднего разрешения. Число таких цветов не может

превышать двух. Обычный режим цветной графики позволяет использовать полную 4-х цветную палитру, в то время как при использовании возможностей построения изображений на основе синтеза символов количество цветов в палитре уменьшается до двух.

В тех случаях, когда таблица рисунков символов, формируемая пользователем, не применяется для построения изображений в графическом режиме, можно использовать специальные символы из расширенного набора. Это может найти применение при имитации символов текстового режима, при построении специальных шаблонов или алфавитов. Так, это наиболее простой и эффективный способ генерации других национальных алфавитов - греческого, славянского (кириллица), японского (кана). Символика арабского языка и иврита также вполне представима, однако, здесь возникают другие сложности - сложности, связанные с особенностями письма (справа-налево).

Приложение 9.1. Текст программы генерации изображений (Паскаль).

```
module listing_9_1;
type
 word_bits_type       =  set of 0..15;
high_res_pixel_type   = (off,on);
medium_res_pixel_type =(color_0,color_1,color_2,color_3;
 graphics_screen_type=
   record
     even_pixel : array [0..99,0..39] of word_bits_type;
     filler     : array [1..192] of byte;
     odd_pixel  : array [0..99,0..39] of word_bits_type;
   end;
var [static]
 graphics_screen_pointer  :  ads of graphics_screen_type;
 general_screen_pointer   :  adsmem;
 row,column               :  integer;
value
 graphics_screen_pointer.s := #B800;
 graphics_screen_pointer.r := 0;
 general_screen_pointer.s  :=#B800;
 general_screen_pointer.r  :=0;
{=================================================
============}
```

ТЕКСТ РАЗОБРАТЬ НЕЛЬЗЯ

```
     graphics_screen_pointer ^ .odd_pixel [(row-1) div 2,column div 16]
else
   work_byte :=
    graphics_screen_pointer ^ .even_pixel [row div 2, column div 16];
work_value := column mod 16;

if color = on then
  work_byte := work_byte + [work_value]
else
  work_byte := work_byte - [work_value];

if odd (row) then
  graphics_screen_pointer ^ .odd_pixel [(row-1) div 2, column div 16]
    := work_byte
else
  graphics_screen_pointer ^ .even_pixel[row div 2, column div 16]
    := work_byte;
end;

{=================================================
=====}

procedure set_medium_res_pixel (color : medium_res_pixel_type);

  var [static]
    work_byte : word_bits_type;
    work_set0 : set of 0..15;
    work_set1 : set of 0..15;

begin

 if odd (row) then
   work_byte :=
 graphics_screen_pointer ^ .odd_pixel [(row-1) div 2, column div 8];
 else
   work_byte :=
 graphics_screen_pointer ^ .even_pixel [row div 2, column div 8];

 work_set0 := [(column mod 8) * 2];
 work_set1 := [(column mod 8) * 2 + 1];

 if color in [color_2,color_3] then
   work_byte := work_byte + work_set0
```

```
 else
   work_byte := work_byte - work_set0;

 if color in [color_1,color_3] then
   work_byte := work_byte + work_set1
 else
   work_byte := work_byte - work_set1;

 if odd (row) then
   graphics_screen_pointer ^ .odd_pixel [(row-1) div 2, column div 8]
    :=work_byte
 else
   graphics_screen_pointer ^ .even_pixel [row div 2, column div 8]
    := work_byte;
end;

{=================================================
=====}

procedure clear_screen_graphics;
  var [static]
    i : word;
  begin
    for i := 0 to 16383 do
      general_screen_pointer ^ [i] := 0;
  end;

{=================================================
=====}

end.
```

ГЛАВА 10. КЛАВИАТУРА

[Оглавление]

Клавиатура - это одно из основных звеньев взаимодействия человека и компьютера. В этой главе мы подробно рассмотрим работу клавиатуры Персонального компьютера фирмы "ИБМ" и методы управления этой работой. Клавиатура IBM/PC имеет ряд интересных особенностей, которые мы сейчас изучим.

10.1. Основные принципы функционирования клавиатуры

На мой взгляд, один из наиболее впечатляющих моментов конструкции IBM/PC - это способ работы с клавиатурой. Выбранный подход в равной мере прост и элегантен и программистам необходимо знать две вещи, связанные с вводом информации от клавиатуры, следующие из этого конструктивного подхода.

На электронном уровне клавиатура IBM/PC представляет собой небольшой самостоятельный компьютер. Внутри блока клавиатуры размещается микропроцессор 8048 фирмы "Интел", который выполняет задачу слежения за нажатиями клавиш и передачи их состояния. Микропроцессор 8048 выполняет различные функции, включая самодиагностику (после включения питания компьютера), проверку нажатия клавиш и противодребезговую защиту (что не позволяет воспринимать одну нажатую клавишу как две).

Микропроцессор 8048 позволяет также буферизовать до 20 нажатий клавиш, если центральный процессор не может их принять сразу. Обычно этот буфер совершенно пуст, поскольку ситуации, когда центральный процессор не успевает отреагировать на запрос клавиатуры.

Вы, вероятно, слышали предупреждающий звуковой сигнал, когда нажатие Вами клавиш опережало возможности программы по их вводу. Это не связано с заполнением буфера клавиатуры. Процедура ввода с клавиатуры системы BIOS в ПЗУ имеет собственный буфер и звуковой сигнал раздается в случае заполнения этого буфера. Буфер клавиатуры может содержать 20 символов, а буфер системы BIOS - только 15. По мере изучения клавиатуры мы рассмотрим как код нажатой клавиши попадает сначала в буфер клавиатуры, затем в буфер BIOS и, наконец, в программу.

Блок клавиатуры не связывает с клавишами никаких конкретных значений. Вместо этого, блок клавиатуры идентифицирует по ее номеру или коду сканирования. Все клавиши имеют коды сканирования от 1 до 83. На рис. 10.1. показаны коды, соответствующие всем клавишам.

При нажатии клавиши блок клавиатуры передает ее код сканирования центральному процессору. Когда клавиша отпускается, клавиатура снова передает ее код, но увеличенный на 128 (или шестнадцатиричное значение 80). Таким образом, имеются различные коды для нажатия и освобождения клавиш.

Для работы с клавиатурой используются порты и прерывания. Когда выполняется какое-либо действие с клавишей (нажатие или освобождение), процессор клавиатуры обнаруживает его и запоминает в своем буфере. Затем, процессор клавиатуры формирует прерывание с номером 9. В ответ на прерывание

служебная процедура системы BIOS в ПЗУ считывает код сканирования клавиши из порта клавиатуры (порт номер 96) и затем пересылает в порт клавиатуры команду очистить буфер процессора клавиатуры. Если системный блок не реагирует на прерывания клавиатуры, то коды сканирования накапливаются в буфере процессора клавиатуры, хотя при нормальной работе этого не должно происходить. Специальный код сканирования 255, шестнадцатиричное значение FF, используется блоком клавиатуры, для сообщения, что его буфер заполнен.

Поскольку информация о нажатии клавиши поступает в системный блок от клавиатуры через порт, любая программа, имеющая доступ к порту, может непосредственно общаться с клавиатурой. На практике, однако, это неприменимо, так как блок клавиатуры вырабатывает еще и прерывания, которые обрабатываются системой BIOS в ПЗУ. Однако, просто из любопытства, можно написать небольшую программу на Бейсике, читающую порт клавиатуры и сообщающую, что она там находит. Ниже приведен фрагмент этой программы.

Поскольку эта программа состязается с системой BIOS за получение информации от клавиатуры, ее работа несколько хаотична. Но если запустить эту программу и нажимать на клавиши, она все же зарегистрирует несколько кодов сканирования.

```
100  REM  Программа на Бейсике, пытающаяся считывать
          коды сканирования клавиатуры
110  X=INP(96)  'чтение порта клавиатуры
120  THROW.AWA4$ = INKEY$ 'отбросить любые коды,считан-
                    системой BIOS
130  IF X=0 THEN GOTO110 'если данных нет - продолжать
                    ожидание
140  PRINT
150  PRINT "Код сканирования клавиатуры"; XMOD128;'сооб-
                                щить код
160  IF X > 128 THEN PRINT "клавиша осбождена";
170  IF X > 129 THEN PRINT "клавиша нажата";
180  GO TO 110
```

Клавиатура IBM/PC выполняет еще и функцию повторения клавиши. Блок клавиатуры следит за тем, сколько времени клавиша остается нажатой и формиирует сигнал повторения. Функция повторения распространяется на все клавиши блока клавиатуры.

Процедуры системы BIOS в ПЗУ могут распознавать отличие повторных нажатий клавиши от повторения сигнала удерживаемой в нажатом состоянии клавиши, путем анализа кодов сканирования

освобожденной клавиши. Если для одной и той же клавиши получены два кода нажатой клавиши и между ними не было кода освобожденной клавиши, значит клавиша удерживается процедурами системы BIOS для подавления функции повторения тех клавиш, которым она не нужна, таких как, например, клавиша смещения (shift).

Теперь Вам должны быть понятны основы того, как IBM/PC работает с клавиатурой. Блок клавиатуры сообщает, что происходит с клавиатурой: какие клавиши нажимаются, какие освобождаются и, через определенный промежуток времени, какие удерживаются в нажатом состоянии. Система BIOS в ПЗУ интерпретирует получаемую от блока клавиатуры информацию, придавая ей определенный смысл. Таким образом, имеет место очень ясное и четкое разделение труда. Блок клавиатуры занимается физической стороной, механизмом функционирования клавиатуры, а программы системы BIOS в системном блоке выполняют все логические операции по интерпретации действий клавиатуры.

Большинство пользователей даже не знает, что клавиатура IBM/PC передает сигналы не только о нажатии, но и об освобождении клавиши, поскольку такая информация обычно скрыта от пользователя. Ее можно увидеть, запустив тест клавиатуры из набора диагностических программ IBM/PC. Если присмотреться, можно заметить, что изображение на экране меняется при каждом нажатии, освобождении клавиши или когда клавиша удерживается в нажатом состоянии достаточно долго для того, чтобы начала действовать функция повторения.

10.2. Определение смысла нажатых клавиш

Служебная процедура системы BIOS для прерывания 9 или прерывания клавиатуры определяет смысл каждого действия с клавиатурой. Это включает слежение за состоянием смещения кодов и преобразование нажатий клавиши в их значения, будь то буквы алфавита или сигналы функциональных клавиш. Ниже мы будем рассматривать эту функцию подробнее.

Часть щадачи обслуживания клавиатуры со стороны системы BIOS заключается в слежении за всеми возможными состояниями смещений. Состояния смещений довольно часто вызывают путаницу, поскольку клавиатура обычной пишущей машинки, с котокой большинство из нас начинало свое знакомство с клавишами, имеет всего одну клавишу смещения, Клавиатуры многих кимпьтеров, в том числе и IBM/PC, имеют три типа смещений.

Во-первых, имеются обвчные функции клавиш, без всякого смещения (например, для получения букв в нижнем регистре).

Затем имеется обычное смещение, такое же как у пишущих машинок, которое служит для получения букв верхнего регистра и, как правило, всех знаков, нанесенных на верхнюю часть клавиши. Кроме того, имеются еще два вида смещения, изменяющее смещение ("ALT") и управляющее смещение ("CTRL"). Эти два специальных вида смещения используются аналогично обычному смещению в том смысле, что просто "a" не совсем то же самое, что "A" в верхнем регистре, и Ctrl-A и Alt-A также имеют собственные значения. (Некоторые читатели могут считать клавишу "ESC" еще одной клавишей смещения. Это связано с тем, что в некоторых старых терминалах компьютеров, особенно таких, которые использовались подобно клавишам смещения. Однако, при использовании клавиши "ESC" она не удерживалась в нажатом состоянии, подобно клавишам смещения, - сначала нажималась клавиша "ESC", а после нее какая-либо другая клавиша. Оба этих нажатия рассматривались как специальная последовательность. Такое действие нельзя рассматривать как операцию смещения. Обычно, клавиша "ESC" подобным образом в IBM/PC не используется.)

Допустимыми считаются не все сочетания обычных клавиш и клавиш смещения. Если ввести одну из недопустимых комбинаций, то система BIOS ее просто проигнорирует, как будто ничего не произошло. Несколько позже мы перечислим все допустимые комбинации смещений клавиш, чтобы Вы могли воспользоваться ими в своих программах.

На клавиатуре IBM/PC имеется четыре специальных "переключательных" клавиши, которые действуют в качестве тумблера "включения-отключения" для определенных функций. Эти клавиши Insert (вставка), Caps-Lock (закрапление заглавных букв), Num-Lock (закрепление цифровой клавиатуры) и клавиша, которой практически не пользуются, Scroll-Lock (закрепление "роллинга"). Две из этих клавиш, Caps-Lock и Num-Lock, являются частью механизма смещения, а оставшиеся две управляют определенными специальными режимами работы.

Информация о состоянии этих четырех переключательных клавиш и том, удерживается ли в данный момент в нажатом состоянии одна из клавиш смещения, хранится в специально выделенной для этих целей области в нижних адресах оперативной памяти. Вся эта информация хранится в двух байтах с адресами 1047 и 1048 (или 417 и 418 шестнадцатиричных). Во время работы IBM/PC, в этих двух байтах сохраняются все состояния смещения и состояния переключателей. Емтественно, что после выключения компьютера эти байты сбрасываются в нормальное состояние.

Программа на Бэйсике, листинг которой 10.1 приведен в конце этой главы, отображает эти два байта и демонстрирует

воздействие на них нажатий клавиш смещения и переключателей. Описание смысла отдельных битов приведено ниже.

Поскольку клавиши смещения действуют только в нажатом состоянии, система BIOS следит за их нажатиями и освобождениями и вносит необходимые изменения в интерпретацию
обычных клавиш. Система BIOS следит за кодами сканирования при освобождении только для клавиши смещения, так как для всех остальных клавиш эта функция не имеет смысла.

Когда система BIOS получает код сканирования обычной клавиши, она проверяет все варианты смещения и преобразует этот код в соответствующее значение, которое может быть как символьным кодом ASCII, так и кодом специального назначения.

Процедура обслуживания клавиатуры системы BIOS следит не только за состояниями смещения. Состояние клавиатуры хранится в первых тринадцати битах указанных выше двух байт по адресу 1047. Ниже приведена таблица значений этих битов, изменения которых можно проследить с помощью программы 10.1:

Байт	Бит	Смысл	Значение, когда бит равен 1
1	1	Insert	состояние активное
1	2	Caps-Lock	состояние активное
1	3	Num-Lock	состояние активное
1	4	Scroll-Lock	состояние активное
1	5	Alt	клавиша нажата
1	6	Ctrl	клавиша нажата
1	7	смещение слева	клавиша нажата
1	8	смещение справа	клавиша нажата
2	1	Insert	клавиша нажата
2	2	Caps-Lock	клавиша нажата
2	3	Num-Lock	клавиша нажата
2	4	Scroll-Lock	клавиша нажата
2	5	Состояние захвата	активное состояние
2	6	(не используется)	(после Ctrl-Num-Lock)
2	7	(не используется)	
2	8	(не используется)	

В этой таблице нашел отражение ряд интересных моментов. Во-первых, можно отметить параллельность использования первых четырех битов обоих байтов для клавиш-переключателей. В первом байте отражается текущее состояние клавиши, а во втором байте указывается нажатие соответствуюших клавиш* Во-вторых. обратив внимание на биты, соответствующие нажатию обычных клавиш смещения, расположенные в правой и в левой частях клавиатуры, распознаются по отдельности. Для такого разделения трудно придумать разумное применение, но тем не менее оно имеет место.

Кроме того, система BIOS следит за состоянием вставки, управляемым переключателем. Эта функция обычно игнорируется программами, которые используют клавишу Iusert, например, текстовыми прогцессорами. Обычно. все программы, использующие эту клавишу, сами следят за режимом вставки. Практического значени[это не имеет, но нам показалось любопытным отметить и такой факт.

И, наконец, обратите внимание на бит состояния захвата, который устанавливается, когда процедура обслуживания клавиатуры системы BIOS обнаруживает комбинацию клавиши Ctrl-NumLock. Эта комбинация используется системой BIOS для управления состоянием захвата, которое программам обнаружить трудно, поскольку в этом состоянии программы не выполняются. В состоянии захвата система BIOS выполняет короткий цикл, ожидая нажатия клавиши, которое выводит систему из этого состояния. В этом состоянии распознаются и обслуживаются все прерывания. Если прерывание поступит от клавиатуры (при нажатии обычной клавиши), то состояние захвата завершается, код нажатой клавиши отбрасывается и управление возвращается той программе, которая выполнялась до установления состояния захвата. Если прерывание имело другую природу (номер, прерывание от дисков), то оно обслуживается, но состояние захвата сохраняется пока не будет нажата клавиша.

Комбинация клавиш Ctrl-NumLock - это один из четырех особых случаев, отслеживаемых процедурой системы BIOS. Когда процедура обслуживания клавиатуры обрабатывает коды, принимаемые от клавиатуры, она постоянно проверяет появление одной из четырех особых комбинаций, которые обслуживаются специальным образом.

Фактически, эти четыре комбинации рассматриваются как команды, по которым должно начинаться выполнение определенных служебных процедур. Эти четыре комбинации: Ctrl-NumLock, PrtSc, Ctrl-Alt-Del и Ctrl-Break.

Как мы уже видели, комбинация Ctrl-NumLock используется для приостановки работы компьютера. Процедура обслуживания клавиатуры не позволяет компьютеру продолжать выполнение

любой программы, пока приостановка не будет отменена нажатием клавиши.

Команда распечатки экрана или PrtSc предназначена для вывода содержимого экрана на устройство печати. Поскольку эта операция выполняется на самом примитивном уровне, она не может использовать возможности ДОС, такие как пересылка сообщения в линию связи вместо устройства печати. Процедура распечатки экрана доступна и Вашим программам на уровне прерываний, что будет показано в главе 11.

Комбинация клавиш Ctrl-Alt-Del используется для перезапуска или для перезагрузки системы. Когда встречается такая комбинация, процедура обслуживания клавиатуры передает управление программе проверки системы и запуска работы операционной системы.

Эту комбинацию клавиш можно использовать в любой момент времени для перезапуска системы, если произошло какое-либо отклонение от нормального функционирования. Однако, как можно заметить, эта комбинация действует не всегда. В некоторых случаях IBM/PC необходимо бывает выключить, а затем снова включить и уже после этого пытаться ее перезапустить. Для чего это бывает необходимо? Комбинация Ctrl-Alt-Del срабатывает всегда, когда работает процедура обслуживания клавиатуры. Это условие может оказаться невыполненным в двух случаях. Первый и наиболее распространенный связан с запретом прерываний. Из главы 3 Вам должно быть известно, что прерывания запрещаются машинной командой CLI и не обслуживаются пока не будет выдана команда STI. Обычно команда STI следует за командой CLI через очень короткий промежуток времени. Однако, если программа по ошибке, оставила прерывания запрещенными, то нажатие клавиш Ctrl-Alt-Del не приведет ни к какому результату. Другая причина нарушения работы процедуры обслуживания клавиатуры может быть связана с изменением значения вектора прерывания от клавиатуры. Если изменить ячейки памяти с 36 по 39, в которых находится вектор прерывания от клавиатуры, тогда клавиатура не будет работать, в том числе и клавиши Ctrl-Alt-Del. Естественно, что прогармма запуска после включения питания восстанавливает все вектора прерываний.

Комбинация Ctrl-Break предназначена для прерывания текущей операции. В отличие от остальных трех специальных комбинаций она может отрабатываться программно. Прерывание номер 27, шестнадцатиричное значение 1B, зарезервировано для процедуры, которая должна вызываться при нажатии клавиш Ctrl-Break. Если программа хочет использовать это средство, она должна занести адрес процедуры обслуживания прерывания в ячейки вектора прерывания 27 - ячейку с адресом 108,

шестнадцатиричное значение 6С. Эту возможность можно исследовать, нажимая клавиши Ctrl-Break при работе с интерпретатором БЕЙСИКА или редактором Edlin, поставляемым в составе ДОС. Если программа не устанавливает адрес процедуры обслуживания прерывания в ячейку 108, то система BIOS совместно с ДОС прерывают выполнение текущей программы или файла пакетной обработки. Управление возвращается ДОС.

Поскольку блок клавиатуры сообщает обо всем, что происходит с клавишами и процедуры системы BIOS в ПЗУ интерпретируют любые действия, выполняемые с клавиатурой, Ваша программа может следить за всеми действиями, производимыми с клавиатурой. Программам редко требуются какие-либо сведения помимо тех, которые сообщаются процедурами обслуживания клавиатуры системы BIOS в ПЗУ, поэтому не обеспечена специальная возможность, позволяющая программе определить какие в точности действия выполнялись с клавиатурой.

Если Вам все-таки необходимо более точно знать, что происходит с клавиатурой, Ваши программы могут подменить вектор прерываний для клавиатуры, который расположен по адресу 36, шестнадцатиричное значение 24. Если Вы разрабатываете собственную процедуру обслуживания прерываний от клавиатуры, то она может либо выполнять абсолютно все функции обслуживания, либо служить пред-процессором для стандартной процедуры системы BIOS. Предпроцессор может извлекать любую информацию, которая Вам необходима, и передавать управление стандартной процедуре для выполнения обычных действий.

10.3.Символьные клавиши

Коды сканирования клавиши преобразуются системой BIOS в расширенные коды ASCII, состоящие из всез 256 возможных байтовых комбинаций, кроме одной (байт с нулевым значением). В этот набор кодов входят обычные символы, набор которых описывается в различных руководствах по IBM/PC. Эти коды можно формировать либо с помощью нажатия обычных клавиш (например, клавиши А для "А" и таек далее) или путем использования клавиатуры и клавиши Alt.

Для формирования кодов ASCII по второму методу нужно, удерживая клавишу "Alt" в нажатом состоянии, вводить числовое значение кода ASCII. Значение должно вводиться в десятичном виде (от 1 до 255) и можно использовать только числовые клавиши в правой части клавиатуры (но не числа в верхнем ряду средней части клавиатуры). При использовании клавиши "Alt"

процедуры системы BIOS работают особым образом, поскольку несколько нажатий клавиш должны интерпретироваться как один символ. Пока нажата клавиша "Alt" можно нажиматьсколько угодно цифровых клавиш. Когда клавиша "Alt" будет наконец освобождена, будет сформирован код символа ASCII, соответствующий введенному числовому значению. Если ввести слишком большое число, используется его значение по модулю 256. Например, если ввести Alt-1000, то будет сгенерирован символ CHR$(232). Деление 1000 на 256 дает остаток 232, поэтому и генерируется символ с кодом CHR$(232).

Единственное значение в коде ASCII, которое невозможно ввести с клавиатуры - это нулевое значение или CHR$(0). На это имеется несколько причин (код 0 определен в системе кодов ASCII как пустой символ, который должен игнорироваться), однако, главная причина заключается в использовании нуля системой BIOS для указания на наличие второго набора символов, специальных символов. Хотя некоторые руководства по IBM/PC указывают, что нулевой код генерируется нажатием клавиши Ctrl-2 или с помощью клавиши "Alt", это неверно. Если удерживать клавишу "Ctrl" в нажатом состоянии и нажать цифровую клавишу "2" будет сформирован один из специальных символов и этот символ должен интерпретироваться как CHR$(0) из набора кодов ASCII. Это не совсем то же самое, что формирование действительного кода ASCII CHR$(0). Использование клавиши "Alt" тоже не даст желаемого результата, будете ли Вы нажимать клавиши Alt-0 или пытаться "обмануть" систему, вводя эквиваленты по модулю 256, например, Alt-256 или Alt-512.

Специальные символы используются для обозначения специальных клавиш, таких как "Home", "End" и десять функциональных клавиш (фактически, как Вы скоро увидите, имеется сорок функциональных клавиш).

Специальные символы позволяют использовать специальные клавиши, такие как функциональные клавиши, не используя ни один из 256 символов расширенного кода ASCII.

Механизм кодирования, который система BIOS в ПЗУ использует для указания, какой символ был введен с клавиатуры (и определение того, обычный это символ или специальный), работает с двумя байтами. Если первый из двух байтов не нулевой, то введен символ расширенного кода ASCII и этот символ хранится в первом байте. Есоли же первый байт нулевой, то с клавиатуры введен специальный символ и его код хранится во втором байте. Здравый смысл подсказывает, что можно было бы использовать один бит второго байта для обычных и специальных символов. Это было бы проще и позволило бы вводить нулевой код ASCII.

В то время как в расширенном наборе ASCII 256 кодов (255 из них можно ввести с клавиатуры), специальных кодов столько, сколько требуется для выполнения всех предусмотренных клавиатурой IBM/PC функций. Например, сорок кодов выделено для функциональных клавиш (десять обычных и еще тридцать для трех возможных смещений - обычного и с помощью клавиш "Alt" и "Ctrl"). Ниже приведена таблица специальных кодов и комбинаций клавиш, которыми они вырабатываются. Вы конечно заметите определенную хаотичность этих кодов. Некоторые комбинации с клавишей "Alt" разрешены, а другие нет, причем без какой-либо видимой причины. То же можно сказать и о комбинациях с клавишей "Ctrl".

Значение специального кода	Клавиши, с помощью которых он формируется
3	Ctrl-2 (что должно соответствовать коду CHR$(0) или ASCII NULL)
15	обратная табуляция (shift-tab)
16-25	от Alt-a до Alt-p (верхний ряд букв)
30-38	от Alt-A до Alt-L (средний ряд букв)
44-50	от Alt-Z до Alt-M (нижний ряд букв)
59-68	от F1 до F10 (функциональные клавиши)
71	клавиша "Home" (курсор в верхний левый угол экрана)
72	клавиша "Курсор вверх"
73	клавиша "PgUp" (страница вверх)
75	клавиша "Курсор влево"
77	клавиша "Курсор вправо"
79	клавиша "End" (конец)
80	клавиша "Курсор вниз"
81	клавиша "PgDn" (страница вниз)
82	клавиша "Ins" (вставка)
83	клавиша "Del" (удалить)
84-93	от смещение-F1 до смещение-F10 (функциональные клавиши с обычным смещением)
94-103	от Ctrl-F1 до Ctrl-F10
104-113	от Alt-F1 до Alt-F10
114	Ctrl-PrtSc
115	Ctrl-Курсор влево
116	Ctrl-Курсор вправо
117	Ctrl-End
118	Ctrl-PgDn
119	Ctrl-Home

120-131	от Alt-1 до Alt-= (верхний ряд клавиш)
132	Ctrl-PgUp

Как Вы могли заметить, эта таблица не слишком регулярна. Вы, наверное, заметили, что имеется сорок комбинаций с функциональными клавишами - каждая из клавиш в одном из четырех смещенных состояний (обычном, смещенном, с клавишей "Ctrl" и с клавишей "Alt").

Имея в своем распоряжении все эти коды, любая программа не будет испытывать недостатка кодов специального назначения. Обычно, для любой программы хватает функциональных клавиш и нескольких клавиш специального назначения, таких как "Home" и клавиши управления курсором. Однако, если потребуются дополнительные клавиши, Вы сможете их найти в этой таблице.

Доступ к этим специальным кодам символов требует изменения определенных особых методов. В следующих двух разделах мы рассмотрим как это делается.

10.4. Процедура обслуживания клавиатуры в системе BIOS

Рассмотрим теперь служебные процедуры системы BIOS, позволяющие осуществлять доступ к клавиатуре.

Процедуры обслуживания клавиатуры в системе BIOS вызываются с помощью прерывания номер 22, шестнадцатиричное значение 16. Таких процедур всего три, поскольку не слишком много функций можно запросить для клавиатуры.

Первая процедура, имеющая код 0, возвращает очередной принятый от блока клавиатуры символ. Эта процедура возвращает управление вызвающей программе только тогда, когда будет введен символ, так что программе не приходится проверять, был ли действительно введен символ. Код обычного символа возвращается в регистре AL, но если содержимоое AL равно нулю, то в регистре AH находится код специального символа. Эти регистры соответствуют первому и второму байтам, упоминавшимся выше, при рассмотрении кодов специальных символов. Ниже описана логика процесса распознавания получаемых символов:

```
если AL=0 то
 начало
   спец_символ:= истина;
   введенный_символ:= AH
 конец
иначе
 начало
```

```
  спец_символ:= AL
конец;
```

Можно сделать ряд интересных замечаний, касающихся значений, возвращаемых этой процедурой. Если введен обычный код ASCII (то есть, AL не равняется нулю), то в регистре AH содержится код сканирования для нажатой клавиши. Но, если код ASCII был введен с помощью клавиши "Alt", то в регистре AH будет ноль. Этту информацию программа может использовать для определения способа вода символа. Некоторые символы кода ASCII встречаются на клавиатуре IBM/PC дважды.

```
1000 REM Листинг 10.1 - программа отображения битов
1010 REM состояния клавиатуры
1030 GOSUB 2000  'вывод заголовка
1040 GOSUB 3000  'дополнительная информация
1050 GOSUB 4000  'отображение битов клавиатуры
1060 GOSUB 5000  'проверка завершения ввода с клавиатуры
1070 GOTO 1050

2000 REM Подпрограмма вывода заголовка
2010 KEY OFF : CLS : WIDTHS 80 : LOCATE  ,,0
2020 REM
2030 LOCATE 5,1
2040 PRINT " Программы для книги 'Персональный компьютер"
2050 PRINT " фирмы ИБМ', автор Питер Нортон, 1983"
2060 PRINT
2070 PRINT "Программа 10-1: Отображение битов состояния"
2080 PRINT "клавиатуры"
2999 RETURN

3000 REM Подпрограмма отображения вспомогательной информации
3010 LOCATE 11,28
3020 PRINT "Байт 1   Байт 2";
3030 LOCATE 12,28
3040 PRINT "12345678 12345678";
3050 LOCATE 17,10
3060 PRINT "Чтобы увидеть изменение битов состояния нажмите";
3065 PRINT "и удерживайте любую из";
3070 LOCATE 18,15
3080 PRINT "-- Левую или правую клавишу смещения";
3090 LOCATE 19,15
3100 PRINT "--Ctrl,Alt,Num-Lock,Scroll-Lock,Caps-Lock,Ins";
3110 LOCATE 21,10
3120 PRINT "(обратите внимание на влияние смещения и"
```

```
3125 PRINT "Num-Lock yf клавишу Ins) ";
3130 LOCATE 24,10
3140 PRINT "Для выхода в ДОС нажмите любую клавишу ввода";
3999 RETURN

4000 REM Подпрограмма отображения битов состояния клавиатуры
4010 DEF SEG = &H40
4020 CONTROL% = PEEK(&H17)
4030 CHECK% = 128
4040 FOR I% = 1 TO 8
4050   LOCATE 14,27 + I%
4060   IF CONTROL% >= CHECK% THEN COLOR 30,0 ELSE COLOR
7,0
4070   IF CONTROL% >= CHECK% THEN PRINT "1"; ELSE PRINT "0";
4080   IF CONTROL% >= CHECK% THEN CONTROL% = CONTROL%-
CHECK%
4090   CHECK% = CHECK / 2
4100 NEXT I%
4110 CONTROL% = PEEK(&H18)
4120 CHECK% = 128
4130 FOR I% = 1 TO 8
4140   LOCATE 14,36 + I%
4150   IF CONTROL% >= CHECK% THEN COLOR 30,0 ELSE COLOR
7,0
4160   IF CONTROL% >= CHECK% THEN PRINT "1" ELSE PRINT "0"
4170   IF CONTROL% >= CHECK% THEN CONTROL% = CONTROL%-
CHECK%
4180   CHECK% = CHECK% / 2
4190 NEXT I%
4999 RETURN

5000 REM Подпрограмма ожидания завершения работы
5010 K$ = INKEY$
5020 IF LEN(K$) = 0 THEN RETURN 'ожидание ввода с клавиатуры
5030 IF (LEN(K$) = 2) AND (CHR$(82) = MID(K$,2,1)) THEN
5035 RETURN  'Нажата клавиша Ins
5040 CLS : LOCATE ,,1
5999 SYSTEM

9999 REM Конец программы 10-1
```

ГЛАВА 11. ДОПОЛНИТЕЛЬНЫЕ СРЕДСТВА - АДАПТЕР СВЯЗИ, ДИНАМИК И ПРОЧЕЕ

[Оглавление]

В этой главе мы рассмотрим доплнительные средства, представляемые IBM/PC, некоторые их низ более сложны, другие менее сложны, но все сложности не таковы не таковы, чтобы посвящать каждому из этих средств самостоятельную главу. Здесь мы рассмотрим устройство печати, асинхронный адаптер связи, интерфейс накопителя на кассетной магнитной ленте, динамик и ряд полезных сведений о системе BIOS в ПЗУ.

11.1. Асинхронный адаптер связи

Асинхронный адаптер связи позволяет IBM/PC общаться с внешним миром с помощью стандартного метода, известного под названием RS-232. В персональных компьютерах адаптер RS-232 обычно используется для двух целей.

Первая цель использования RS-232 собственно для связи, по телефонным линиям. При этом адаптер RS-232 подключается к модему (или модулятору-демодулятору), который выполняет преобразование сигналов компьютера в телефонные сигналы. Модем, в свою очередь, подключается к телефонной линии. На другом конце линии должен находиться другой модем, который может быть подключен к чему-нибудь. Это может быть все что угодно, от универсального компьютера до другого персонального компьютера или какого-либо простого устройства, например, устройства печати. Это, нужно сказать, и является нормальным использованием адаптера связи.

Другой способ использования RS-232 значительно проще. Некоторые устройства вывода разработаны в соответствии с протоколом взаимодействия RS-232. Наиболее часто это относится к устройствам печати, особенно посимвольным устройствам. Таким образом, адаптер связи IBM/PC может использоваться просто в качестве дополнительного средства взаимодействия с периферийным оборудованием, таким как устройство печати.

Сделаем небольшое отступление и рассмотрим два основных способа взаимодействия персональных компьютеров, в том числе IBM/PC, с периферийным оборудованием. Эти два способа кратко называют последовательным и параллельным. Протокол RS-232 является "последовательным", поскольку данные передаются по одной линии и биты данных посылаются последовательно, по одному. Параллельная связь реализует другую схему взаимодействия, которая по названию популярных устройств печати фирмы "центроникс", получила название параллельного интерфейса типа "Центроникс" или, для краткости, просто параллельного интерфейса. При параллельном взаимодействии данные передаются побайтно и этот интерфейс включает столько линий связи, сколько необходимо чтобы передавать все эти биты

одновременно.

Параллельный интерфейс приспособлен для непосредственной связи компьютера с внешними устройствами, в частности, что данные могут передаваться быстрее, поскольку за один цикл передачи пересылается сразу целый байт. Параллельный интерфейс типа "центроникс" используется в качестве стандартного адаптера связи с устройством печати в IBM/PC. Последовательный интерфейс работает медленнее, однако, он имеет ряд специальных возможностей для организации взаимодействия с удаленными объектами. Все эти возможности не нужны при локальном использовании последовательного интерфейса, для взаимодействия с устройством печати. Однако, это может оказаться удобным способом организации связи между компьютером и устройством печати, в частности, еще и потому что многие персональные компьютеры предыдущего поколения не имели параллельного интерфейса.

Осуществление связи с удаленным абонентом может оказаться очень сложной задачей, поскольку может возникнуть множество различных неполадок, требуется контроль многочисленных ошибок и необходимо работать с разнообразным оборудованием, подключенным к линии. Однако, служебные процедуры, входящие в систему BIOS IBM/PC для обслуживания адаптера RS-232 достаточно просты и удобны в использовании.

Имеется всего четыре служебных процедуры для связи и все они вызываются с помощью прерывания 20, шестнадцатиричное значение 14: регистр DX используется для указания того, какой адаптер связи должен быть использован, если их больше одного; нулевое значение соответствует первому (и, обычно, единственному) адаптеру. Программы 11-101 и 11-102, входящие в дисковый пакет, прилагающийся к этой книге, содержит ассемблерные интерфейсы подпрограммы и вспомогательные программы на Паскале, позволяющие наиболее полно воспользоваться всеми процедурами связи системы BIOS в ПЗУ.

Первая процедура, с кодом 0, устанавливает четыре стандартных параметра связи: скорость передачи, способ контроля по паритету, количество стоповых битов и длину слова. Эти параметры задают различные вариации организации связи. IBM/PC может использовать самые различные их сочетания и обычно конкретные значения определяются тем оборудованием, которое находится на другом конце линии связи. Коды этих переменных выбираются из регистра AL следующим образом:

Первые три бита задают скорость передачи в битах в секунду. Восемь возможных значений соответствуют довольно большим скоростям передачи, хотя персональные компьютеры наиболее часто используют две скорости - 300 и 1200 бит в секунду. Ниже приведены коды и соответствующие им значения:

Код	Скорость передачи (бит в секунду)
000	110
001	150
010	300 (примерно 30 символов в секунду)
011	600
100	1200 (примерно 120 символов в секунду)
101	2400
110	4800
111	9600

(Издержки организации связи - необходимость передачи битов паритета и стоповых битов - увеличивают число битов, необходимых для передачи символа. Поэтому скорость передачи 300 бит в секунду позволяет передавать примерно 30 символов в секунду, хотя в коде символа в ASCII всего 7 или 8 бит).

Следующие два бита определяют способ контроля по паритету:

Код	Способ контроля
00	отсутствует
01	контроль по нечетности
10	отсутствует
11	контроль по четности

Следующий бит указывает чмсло используемых стоповых битов, являющихся обязательной частью протокола RS-232:

Код	Число стоповых битов
0	1
1	2

Последние два бита параметра определяют длину используемого слова:

Код	Длина слова
10	7 бит (стандартный код ASCII)
11	8 бит (обычная длина слова для компьютеров)

Как правило, выбор всех этих параметров не произволен и определяется характеристиками системы или оборудования, с которыми Вы связываетесь.

Вторая процедура для связи, с кодом 1, используется для пересылки одного байта данных. Этот байт загружается в регистр AL и сигнал завершения возвращается в регистр AH. Если пердача завершилась неудачно, то первый бит в регистре AH будет установлен в единицу, а значение остальных семи бит будет в этом случае таким же, какое описано ниже для процедуры с кодом 3. Таким образом, сравнение содержимого регистра AH с числом 128 позволит определить успешно ли закончилась передача. Как обычно, интерфейсная процедура, входящая в прилагающийся к этой книге пакет, возьмет на себя заботу обо всех этих подробностях.

Третья процедура для связи, с кодом 2, используется для приема байта из линии связи. Эта процедура ожидает завершения операции (которое может заключаться в обнаружении ошибки или условия тайм-аута).

Эта процедура содержит одну из наиболее сушественных ошибок в исходной версии системы BIOS-ПЗУ; если версия BIOS датирована 24.04.1981, то в ней сообщение о тайм-ауте ошибочно передается как сообщение об ошибке по паритету с данными, готовыми для приема. Определить свою версию BIOS Вы можете с помощью отладчика ДОС-DEBUG. Если вызвать DEBUG и ввести
команду:

```
D F000:FFF5 L8
```

то будет отображена дата создания версии BIOS. Более подробно ошибки первой версии ситемы BIOS рассмотрены в главе 6.

Если операция приема закончилась успешно, регистр AH будет обнулен. В противном случае, единицы будут в первом бите регистра AH и в некоторых битах с 4-го по 7-ой, как описывается ниже.

Последняя процедура, с кодом 3, используется для определения полного текущего состояния порта связи (в регистре AH) и его модема (в регистре AL). Часть этой информации сообщается и при выполнении процедур с кодами 1 и 2. Регистры AH и AL устанавливаются следующим образом:

Регистр	Бит	Значение
AH	1-ый	тайм-аут (кроме BIOS версии 24.4.81)
AH	2-ой	сдвиговый регистр передачи пуст

AH	3-ий	буферный регистр передачи пуст
AH	4-ый	обнаружен разрыв связи
AH	5-ый	ошибка в битах обрамления
AH	6-ой	ошибка по паритету (BIOS версии 24.4.81 устанавливается в случае тайм-аута)
AH	7-ой	коллизия
AH	8-ой	данные готовы (BIOS версии 24.4.81 устанавливается в случае тайм-аута)
AL	1-ый	есть сигнал в линии
AL	2-ой	есть вызов
AL	3-ий	модем готов
AL	4-ый	сигнал отбоя посылки
AL	5-ый	есть дельта - огибающая сигнала при приеме
AL	6-ой	есть задний фронт сигнала вызова
AL	7-ой	есть дельта - огибающая модем
AL	8-ой	сигнал отбоя по дельта-огибающей

Вся приведенная выше информация предназначена для того, чтобы позволить специалисту по телекоммуникации начать программировать для IBM/PC.

11.2. Адаптер устройства печати

Система BIOS-ПЗУ включает процедуры поддержки для параллельного адаптера устройства печати. (Различия параллельных и последовательных адаптеров рассмотрены в предыдущем разделе). Эти процедуры проще, чем процедуры обслуживания связи, поскольку проще само устройство печати. Имеется всего три процедуры и используется только шесть бит состояния.

Для доступа к этим трем процедурам используется прерывание 23, шестнадцатиричное значение 17. Регистр DX указывается для указания того, какой адаптер должен использоваться, когда их больше чем один; ноль обозначает первый (и обычно единственный) адаптер. Программы 11-101 и 11-102 в дисковом пакете, прилагающемся к этой книге, включают интерфейсные подпрограммы на Ассемблере и вспомогательные процедуры на Паскале, необходимые для более полного использования этих служебных процедур системы BIOS.

Одна из незначительных ошибок в исходной версии системы BIOS-ПЗУ касается как раз обслуживания устройства печати. Когда команда перевода страницы посылается стандартному устройству печати фирмы "Эпсон", система BIOS отводит на выполнение этой операции несколько меньше времени, чем

фактически может потребоваться устройству. Это происходит в тех случаях, когда команда перевода страницы выдается вблизи верхнего края старой страницы. Во всех версиях BIOS после 24.4.1981 это время увеличено.

Первая процедура с кодом 0, используется для пересылки одного байта устройству печати. Эта процедура очень проста и о ней больше нечего сообщить.

Вторая процедура с кодом 1, используется для сброса устройства печати и определения его состояния. Эта процедура может использоваться для перевода устройства печати в нормальное состояние после того как ему были посланы какие-нибудь специальные коды управления. Информация о состоянии возвращается в регистре AH,причем биты имеют следующие значения:

Бит	Значение
1-ый	устройство печати занято
2-ой	сигнал подтверждения
3-ий	сигнал отсутствия бумаги
4-ый	сигнал выборки
5-ый	сигнал ошибки вывода
6-ой,7-ой	не используются
8-ой	сигнал тайм-аута (для BIOS версии 24.4.1981 это может оказаться ложная тревога)

Третья и последняя процедура с кодом 2 используется для чтения битов состояния, описанных выше, без выполнения сброса. Эта процедура особенно полезна для программ управления устройством печати.

Вот и все, что нужно былосказать об устройстве печати. Это не слишком сложное устройство и, соответственно, просты процедуры его обслуживания.

11.3. Интерфейс кассетного накопителя

Очередная часть ссистемы BIOS-ПЗУ относится к интерфесу кассетного накопителя. Он практически никем не используется и многие, включая и меня, не видят других причин включения его в состав IBM/PC, кроме чисто коньюктурных проблем сбыта. (Более подробно этот вопрос рассматрьивается в главе 2).

Имеется всего четыре простых служебных процедуры для обслуживания кассетного накопителя : дл[чтени[и записи блоков данных и для включения и выключения двигателя. Отсутствуют команды перемотки кассеты: эта операция должна выполняться вручную с пульта управления кассетным накопителем.

Заметим, что имеющиеся команды носят "физический" характер, и совсем нет "логических команд, таких как команда поиска файла. Команды такого рода относятся скорее к уровню операционной системы, чем к уровню системы BIOS - в конце концов, система BIOS должна обеспечивать наиболее примитивный, элементарный уровень обслуживания, на базе которого могут строиться логические процедуры более высокого уровня. К сожалению, ДОС совершенно не поддерживает интерфейс кассетного накопителя, так что с ним можно работать только из Бейсика или на примитивном уровне системы BIOS. Это существенно ограничивает возможности тех, кто хотел бы серьезно использовать кассетный накопитель.

Для доступа к четырем процедурам обслуживания кассетного накопителя используется прерывание 21, шестнадцатиричное значение 15. Программы 11-101 и 11-102 в дисковом пакете, прилагающемся к этой книге, включают интерфейсные подпрограммы на ассемблере и вспомогательные процедуры на Паскале, необходимые для более полного использования этих служебных процедур системы BIOS.

Первая процедура, с кодом 0, включает двигатель кассетного накопителя. Эта процедура возвращает уцправление немедленно, не ожидая пока двигатель раскрутится - это необходимо учитывать.

Вторая процедура, скажем 1, выключает двигатель.

Третья процедура, с кодом 2, используется для чтения блоков данных с кассеты. Данные передаются блоками по 256 байт и за один раз можно считывать несколько блоков. Если в процессе передачи данных будект обнаружена ошибка, сообщение об этом будет передано в регистре АН.

Код ошибки	Значение
1	обнаружена ошибка циклического избыточного кода (CRC)
2	утрачены некоторые реквизиты данных (использующиеся для записи размещения и информации о битах)
4	данные не найдены

Четвертая процедура, с кодом 3, использщуется для записи блоков данных на кассету.

На этом мы завершаем изучсение служебных процедур системы BIOS-ПЗУ, предназначенных для поддержки периферийных

устройств, подключаемых к IBM/PC, но этим не исчетпывается ни система BIOS, ни оборудование IBM/PC. В следующем разделе мы рассмотрим все остальное. Но прежде чем двигаться дальше, нужно сообщить еще одну подробность, касающуюся интерфейса кассетного накопителя.

Интерфейс кассетного накопителя IBM/PC разрабатывался только для подключения кассетного накопителя к IBM/PC и он практически не используется ни в одном компьютере. некоторые умные люди приспосабливают этот интерфейс для соединени[IBM/PC с другими устройствами. Одно из возможных использований - подключение специализированных контроллеров буферированного приема/передачи.

11.4. Дополнительные процедуры системы BIOS

Помимо тех процедур, которые мы уже рассмотрели, система BIOS включает еще ряд интересных и полезных служебных процедур. Программы 11-101 и 11-102, входящие в дисковый пакет, прилагаемый к этой книге, включают интерфейсные подпрограммы на ассемблере и вспомогательные процедуры на Паскале, позволяющие наиболее полно использовать эти служебные процедуры системы BIOS.

Первая из этих процедур - это процедура распечатки содержимого экрана. Она обычно вызывается процедурой обслуживания клавиатуры системы BIOS-ПЗУ, когда та обнаруживает нажатие клавиши "PrtSc". Распечатка экрана выполняется программой обслуживания прерывания номер 5 и это позволяет программно вызывать ту же операцию, запрашивая прерывание номер 5. Процедура обслуживания клавиатуры, обнаружив нажатие клавиши "PrtSc", просто вызывает прерывание 5.

Пользующиеся интерпретатором Бейсика замечали, что при работе с Бейсиком клавиша "PrtSc" не действует. Однако, имеется простой способ вызова программы распечатки экрана из программ на Бэйсике. Сам интепретатор изменяет способ обслуживания клавиатуры и здесь ничего изменить нельзя, но можно просто запросить из программы на Бейсике прерывание 5, описанное выше.

Для программ на Бейсике необходима интерфейсная процедура на машинном языке, позволяющая обратиться к операции распечатки экрана, Это очень простая процедура, состоящая всегно имз двух команд, занимающих три байта. На я[зыке ассемблера эти команды записываются так:

```
INT; вызов прерывания номер 5 (распечатка экрана)
  ; (2-х байтная команда)
RET;  взврат в программу на Бейсике (1 байт)
```

Такую простую подпрограмму можно просто записать в память командами POKE и, поскольку длина ее всего три байта, можно использовать числа обычной точности, Следующая короткая программа показывает, как это делается:

```
100 REM Простая программа, позволяющая организовать
110 REM вызов операции распечатки экрана из Бейсика
120 REM Трехбайтная программа на машинном языке
125 REM формируется внутри переменной.

130 HOLD.THE.CODE = 0 'выделяется 4-байтная переменная
140 REM              обычной точности
150 PRINT.SCREEN = VARPTR(HOLD.THE.CODE)' установка
                          указателя
160 REM              на область памяти
170 POKE PRINT.SCREEN + 0,205 '1-й байт подпрограммы -
175 REM              команда прерывания (INT)
180 POKE PRINT.SCREEN + 1,5  '2-й байт - номер преры-
          вания 5
190 POKE PRINT.SCREEN + 2,203 '3-й байт -команда возв-
          рата
200 REM
210 REM Теперь подпрограмма в машинных кодах
220 REM храниться в переменной "hold.the.code"
230 REM и ее адрес храниться в "print.screen"
240 REM
250 CALL PRINT.SCREEN
```

В этой простой программе строки 120-170 записывают подпрограмму в машинных кодах. Эту процедуру достаточно выполнить один раз. Оператор CALL в строке 250 может выполняться тогда, когда необходимо распечатать содержимое экрана. (Интерфейсные подпрограммы для всех процедур системы BIOS и ДОС включены в дисковый пакет, прилагаемый к этой книге, но такой простой интерфейс, как тот, который описан выше, можно оформить непосредственно на Бейсике. Для более сложных процедур обычно требуются законченные интерфейсные подпрограммы на ассемблере, которые могут загружаться из Бейсика процедурой BLOAD).

Следующая из дополнительных процедур системы BIOS - процедура вывода списка оборудования, которая вызывается с помощью прерывания номер 17, шестнадцатеричное значение 11. При первом включении IBM/PC процедуры запуска и самопроверки проверяют, какое оборудование подключено к IBM/PC (в основном, проверка сводится к считыванию позиций переключателей, установленных на системной плате IBM/PC; эти

переключатели должны выставляться определенным образом в зависимости от состава подключенного оборудования. Такой метод не слишком точен, поскольку переключатели могут быть выставлены неправильно).

Ваши программы могут выяснить конфигурацию системы, запрашивая прерывание 17. При выполнении процедуры обслуживания этого прерывания не производится ни новая проверка состава оборудования, ни повторное считывание положений переключателей. Происходит всего лишь пересылка двух байтов, содержащих код конфигурации системы, вызывающей программы в регистрах AH и AL.

Ниже приведена таблица кодов оборудования:

Регистр	Бит	Значение
AH	1-2	число адаптеров устройства печати (от 0 до 3)
AH	3	не используется
AH	4	игровой адаптер подключен или нет
AH	5-7	число адаптеров связи (от 0 до 7)
AH	8	не используется
AL	1-2	число дисководо в без единицы (если установлен 8-ой бит)
AL	3-4	начальный видеорежим (см.таблицу ниже)
AL	5-6	объем памяти на системной плате (см.таблицу ниже)
AL	7	не используется
AL	8	дисководы подключены (если да, см. 1 и 2 биты)

Заметим, что число дисководов, задаваемое первыми двумя битами AL, копируется числом, на единицу меньше действительного числа, так что значение 00 соответствует одному дисководу, 01 - двум и так далее. С помощью такой схемы кодирования указывается наличие до 4-х дисководов. Восьмой бит указывает, имеются ли вообще дисководы в системе. Если этот бит установлен, то используются бва бита счетчика. Проще было бы использовать три бита для кодирования числа дисководов, что позволило бы задавать числа от 0 до 7.

Следующая таблица поясняет значения битов для начального видеорежима:

Биты	Значение

00	(не используется)
01	графический режим, 40 столбцов, черно-белое изображение
10	графический режим, 80 столбцов, черно-белое изображение
11	монохромный режим

Следующая таблица поясняет установку битов для кода объема памяти на системной плате:

Бит	Значение
00	16К
01	32К
10	48К
11	64К

Ваши программы могут использовать эту информацию для настройки своей работы на конфигурацию системы.

Следующая процедура сообщает, какой объем памяти подключен к компьютеру. Имеется ввиду объем рабочей памяти, включающий объем памяти на системной плате (сообщаемый предыдущей процедурой) и дополнительные платы памяти. Для ызова этой процедуры служит прерывание 18, шестнадцатиричное значение 12, а значение, возвращаемое в регистре АХ, равняется числу блоков памяти размером 1К. Такой способ позволяет определить полный объем памяти в один мегабайт.

Значение объема памяти не устанавливается прямой проверкой, а берется из стандартной ячейки памяти. Это позволяет программно изменять объем используемой памяти.

Одно из самых полезных средств для работы с IBM/PC с большим объемом памяти (например, 512К) включает системную программу, которая использует часть памяти как некий очень быстрый накопитель на гибком диске, т.е. так называемый элекронный диск . Такой тип модификации системы может быть очень полезен и начинает использоваться все шире. Для того чтобы иметь возможность выделения физической памяти для какого-нибуть специфического использования, IBM/PC хранит рабочее значение объема доступной памяти в ячейке оперативной памяти, содержимое которой можно изменить. И при любом использовании памяти, например, при ее выделении программам, объем памяти всегда сравнивается с этим значением.

Еще одна интересная особенность ПЗУ не связана с системой BIOS, а относится к способу вызова кассетной системы Бейсика, которая также хранится в ПЗУ. Если значение 18, тогда уравление компьютером передается иетерпретатору Бейсика. Следует иметь в виду, что такая операция уже не позволит снова получить управление Вашей программе и даже операционной системе ДОС.

Еще одна процедура, которая удаляет Вашу программу и ДОС из системы, - это процедура начальной загрузки, вызывающаяся прерыванием 25, шестнадцатиричное значение 19. Эта процедура перезагружает операционную систему с диска, так же как это происходит после нажатия клавиши Ctrl-Alt-Del. Единственное отличие заключается в том, что при нажатии клавиши Ctrl-Alt-Del выполняется диагностика системы, такая же как при включении питания, а при запросе прерывания 25 сразу производится загрузка системы. Это один из самых радикальных способов завершения программы, гарантирующий наличие в памяти новой копии операционной системы.

Следующая процедура - это процедура обслуживания таймера. Она используется для считывания и изменения внутреннего счетчика таймера. Хотя эту процедуру и называют процедурой текущих даты/времени, на самом деле это всего лишь счетчик прерываний таймера. Счетчик увеличивается на единицу при каждом прерывании, которое происходит примерно 18-21 раз в секунду. Вычисление и обновление времени дня и даты - это операция ДОС, которая использует результаты счетчика.

имеется две процедуры обслуживания таймера, вызывающиеся по прерыванию 26, шестнадцатиричное значние 1А, одна, с кодом 0, для считывания показаний таймера, а вторая, с кодом 1, для их установки. Значение счетчика таймера хранится в виде пары двухбайтовых чисел, использующихся, фактически, как одно четырехбайтовое. Эта процедура вносит лишь одно добавление в примитивную процедуру подсчета частов. Когда это происходит, счетчик сбрасывается и факт запоминается. При очередном запросе на считывание значения счетчика передается сообщение о том, что таймер сбрасывался.

Это позволяет ДОС следить за временем дня и сменой дат не вычисляя их постоянно. Когда ДОС необходимо установить время дня, используется описанная процедура для получения значения счетчика и по нему вычисляется время; если будет получено сообщение о сбросе таймера, то ДОС изменит также и дату. Таким образом, ДОС выполняет все эти вычисления только тогда,когда требуется указать время, что упрощает функции процедуры обслуживания прерываний от таймера, которая должна выполняться 18 раз в секунду. Если к ДОС будет хотя бы один запрос времени за день, то проверить это, оставив Ваш

компьютер включенным на 24 часа. К концу Вашей проверки время дня будет по прежнему правильным, а дата ошибочной).

Если Вы будете сами считывать или устанавливать показания таймера с помощью этих процедур системы BIOS, это может помешать правильному отсчету времени и даты в ДОС. ДОС устанавливает значение счетчика таймера таким образом, как будто он был запущен в полночь, после чего ДОС полностью полагаетс[на значение признака сброса таймера. Любая установка или считывание показаний таймера Вашими программами естественно окажет влияние на отсчет времени в ДОС. Если Ваши программы будут только считывать показания таймера, это может привести к перехвату сообщения о сбросе таймера, что нарушит порядок отсчета даты в ДОС.

Теперь мы рассмотрели все процедуры системы BIOS-ПЗУ.

Вы могли обратить внимание на два упущения в составе этих процедур. В то время как все устройства, подключаемые стандартным образом, имеют поддерживающие их служебные процедуры,совершенно не поддерживается игровой адаптер. Это происходит потому, что такая поддержка просто не нужна. Я могу только предполагать почему это произошло; одна из возможных причин может заключаться в том, что форма "ИБМ" не сочла этот адаптер дочтаточно важной составной частью системы, которой требуется какая-либо поддержка на уровне системы BIOS. Возможно, также что игровой адаптер был доставлен к системе на достаточно поздней стадии ее разработки и его поддержка не могла быть включена в систему BIOS, размещенную в ПЗУ.

Еще одна недостающая часть системы BIOS - это набор процедур для обслуживанния встроенного динамика. Почему они не были включены в систему - это для меня загадка. В следующем разделе мы рассмотрим работу динамика и то, как извлекать из него звуки под управлением программы.

11.5. Рецензия звука с помощью динамика

К сожалению, в системе BIOS нет универсальных процедур обслуживания встроенного в IBM/PC динамика. Хотя в системе имеются две подпрограммы, позволяющие извлекать звуки из динамика, но они могут использоваться только внутри самой системы BIOS - это не служебные процедуры общего пользования.

В данном разделе мы поясним работу динамика и поясним основы его использования.Дисковый пакет, прилагающийся к этой книге, содержит служебные подпрограммы на Ассемблере,которые обеспечат Вам доступ ко всем возможностям динамика.

Любой громкоговоритель - будь то часть высоко-качественной системы воспроизведения звука, телефонный

динамик или динамик IBM/PC - работает за счет получения последовательности электрических импульсов, которые вызывают отклонение диафрагмы динамика, вызывающее колебания воздуха, образующие звук. В обычном динамике, поступающие импульсы имеют различную мощность (громкость) и длительность (частоту или высоту тона). Простой динамик IBM/PC не позволяет регулировать громкость и звук образуется просто последовательностью электрических импульсов. Ток к динамику подается и отключается в режиме простого двоичного управления. Частота импульсов тока определяет звуковую частоту, вырабатываемую динамиком. Например, если ток включается и выключается 300 раз в секунду, то динамик генерирует звук частотой 300 герц.

Схемы управления динамиком очень интересно организованы и обеспечивают два способа формирования звука. Импульсы, посылаемые динамику, управляются комбинацией программного сигнала и программируемого таймера. Два способа формирования звука определяются тем, используется таймер или нет.

Сначала рассмотрим как формируется звук без использования таймера. Чтобы заставить динамик звучать программа должна задавать импульсы нужной частоты в течение нужного времени. Хотя это делается на ассемблере, я использовал Паскале-подобные конструкции просто для иллюстрации логики этой операции. Предположим, что необходимо воспроизвести звук с частотой 300 герц, который должен звучать половину секунды. Вот что необходимо сделать:

повторить 150 раз { то есть, 1/2 секунды, при частоте 300 Гц }
начало
импульс отключения динамика
задержка на 1/600 секунды {половина цикла}
{задержка времени выполняется с помощью холостого цикла, повторяющегося нужное число раз}
импульс включения динамика
задержка еще на 1/600 секунды {вторая половина цикла}
конец;

Если Вы внимательно изучите программу на ассемблере BEEP/WARBLE, приведенную в виде листинга 3.2. в главе 3, Вы заметите, что она работает именно таким образом.

Это очень хороший способ воспроизведения звука, но он не позволяет программе ничего делать, кроме генерации звука. Фактически, эта маленькая программа использует весь ресурс микропроцессора 8088, не делая ничего кроме подсчета времени и формирования импульсов для динамика. Для этого есть более совершенный способ, использующий таймер.

Одна из скрытых возможностей IBM/PC - это наличие

программируемого таймера. Он не измеряет никаких промежутков времени, а только подсчитывает импульсы основного тактового генератора системы. Для этого таймера задается число, называемое коэффициентом деления (или просто делителем), и он подсчитывает число импульсов тактового генератора и сравнивает это число с делителем. Когда эти числа сравняются, таймер выдает сигнал и снова начинает подсчет с нуля.

Системный тактовый генератор работает на чистоте 1,19мГц. Так что если таймер запрограммирован на 10000 импульсов, он будет выдавать сигналы примерно 100 раз в секунду. Можно добиться любой частоты импульсов на выходе таймера, подбирая соответствующий делитель. После загрузки делителя схемы динамика запускаются для работы под управлением таймера, после чего выходные сигналы таймера будут управлять частотой формируемого звука, а компьютер может выполнять любую другую работу.

Такой способ управления динамиком позволяет программам работать, пока динамик воспроизводит звук. Таким образом организована работа одной из процедур Бейсика, фоновая музыка или MB. Небходимо заметить, что при формировании звука с помощью таймера, он продолжает звучать до тех пор, пока он не будет отключен той же программой , которая его запустила.

Листинг 3.2. может использоваться в качестве примера формирования звуков без использования таймера. Программа на ассемблере, представленная листингом 11.1., показывает как запустить звучание динамика с помощью таймера.

Приложение 11.1. Текст программы генерации звука с использованием таймера (Ассемблер).

```
a440seg segment 'code'

    assume cs:a440seg

a440  proc   far

    mov    al,0b6h
    out    67,al

    mov    ax,2711
    out    66,al
    mov    al,ah
```

```
        out    66,al

        in     al,97
        or     al,03
        out    97,al

        int    20h

a440  endp

a440seg  ends

        end
```