О.В. Караваева

СИСТЕМНЫЕ ПРОГРАММНЫЕ СРЕДСТВА ДЛЯ УПРАВЛЕНИЯ ВВОДОМ–ВЫВОДОМ.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

УЧЕБНОЕ ПОСОБИЕ

Киров

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

О.В. Караваева

Системные программные средства для управления вводом–выводом.

Лабораторный практикум

Допущено учебно-методическим
объединением вузов
по университетскому
политехническому образованию
в качестве учебного пособия
для студентов, обучающихся
по специальности 230101
«Вычислительные машины,
комплексы, системы и сети»

Киров 2007

Печатается по решению редакционно-издательского совета Вятского государственного университета

УДК 004.424.8(07)
К21

Караваева О.В. Системные программные средства для управления вводом-выводом. Лабораторный практикум: учебное пособие – Киров: Изд-во ВятГУ, 2006. – 114с.

В учебном пособии предлагается цикл из четырех лабораторных работ: изучение архитектуры процессора, работа с прерываниями на языке Ассемблера для управления клавиатурой и монитором, изучение структуры жесткого и логического дисков и работа с системными областями дисков в файловой системе FAT. В практикуме дано описание работы турбо-отладчика, с помощью которого изучается архитектура процессора в реальном режиме. В первой части рассматриваются основные приемы программирования на языке Ассемблера при работе со стандартными символьными устройствами с использованием прерываний DOS и BIOS. Приведены примеры программ. Во второй части дано описание структуры диска и показаны основные приемы работы с дисками на низком уровне в операционных системах DOS, Windows 95/98/ME, Windows NT/2000/XP/2003/Vista.

Пособие рассчитано на студентов, обучающихся по специальности 230101 (220100) «Вычислительные машины, комплексы, системы и сети» при изучении дисциплин "Системное программное обеспечение" и "Операционные системы" Оно может быть полезным студентам других специальностей при знакомстве с использованием языка Ассемблера для управления устройствами ввода-вывода, дисками и файлами.

Редактор Е.Г. Козвонина

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Специалист по направлению «Информатика и вычислительная техника» должен не только изучить теоретические принципы построения и функционирования операционных систем и основы разработки системного программного обеспечения, но также уметь разрабатывать самостоятельно отдельные компоненты операционной системы, или системные утилиты.

Одной из основных задач операционных систем является рациональное управление ресурсами компьютера для того, чтобы повысить эффективность его использования.

Разработчики современных операционных систем стремятся предоставить пользователю максимальные удобства для работы, как на уровне пользователей, так и на уровне разработчиков приложений. Работа на компьютере становится все более доступной для непрофессиональных пользователей, операционные системы совершенствуются, предоставляя пользователям большое количество всевозможных услуг. Но, с другой стороны, операционные системы становятся все более защищенными и закрытыми, а пользователи все более отдаляются от особенностей организации различных операционных систем и тем более от аппаратуры компьютера.

Естественно, для простого пользователя предоставление всевозможных услуг операционной системой и системным программным обеспечением чрезвычайно привлекательно. Для разработки пользовательских приложений можно воспользоваться универсальными системными средствами и больше внимания уделять самой проектируемой задаче.

Проблема возникает, когда необходимо обратиться напрямую к аппаратуре компьютера, например для организации нестандартной работы того или иного устройства. Таких задач достаточно много, и в первую очередь это задачи, организующие защиту программ от копирования или защиту данных от несанкционированного доступа.

Поэтому чрезвычайно важно при изучении дисциплины «Системное программное обеспечение» иметь возможность создавать приложения, позволяющие напрямую обратиться к устройствам ввода-вывода или к системным областям диска.

Решить проблемы, связанные с аппаратурой, невозможно без знания языка Ассемблера. Ассемблер является символическим представлением машинного языка, поэтому он неразрывно связан с архитектурой самого микропроцессора. Язык ассемблера точно отражает все особенности машинного языка, поэтому самая эффективная программа может быть написана только на нем, но в настоящее время это относится только к программам, которые должны обеспечить эффективную работу с аппаратной частью. При программировании на Ассемблере сохраняется полный доступ к аппаратным ресурсам компьютера.

Кроме того, для изучения организации файловых систем на практике необходимо, чтобы операционная система позволяла обращаться к файлам и дискам на уровне прерываний BIOS. Современные операционные системы, и в частности файловые системы с их обособленностью и закрытостью, не позволяют

таких обращений, и даже при работе с дискетой могут возникнуть проблемы. Поэтому при изучении дисциплины «Системное программное обеспечение» предлагается использовать операционную систему Windows98 с файловой системой FAT (FAT12, FAT16, FAT32) для изучения на практике разделов «Управление устройствами», «Управление вводом-выводом» и «Управление файлами и дисками».

Настоящий практикум охватывает две основные формы работы студентов: лабораторные занятия и самостоятельную подготовку к ним.

В соответствии с Государственным образовательным стандартом высшего профессионального образования по направлению «Информатика и вычислительная техника», дисциплина «Системное программное обеспечение» входит в цикл специальных дисциплин и обучение по ней проводится на III курсе. К этому времени студенты уже приобретают необходимые знания по таким дисциплинам, как «Технология программирования», «Математическая логика и теория алгоритмов», «Теория автоматов», «Организация ЭВМ и систем». Общий объем дисциплины 170 часов, который обычно распределяется следующим образом: лекции – 51 час, лабораторные занятия – 34 часа, самостоятельная работа – 81 час. В зависимости от сложности на отдельную лабораторную работу отводится два или четыре часа.

Государственный стандарт по дисциплине «Системное программное обеспечение» содержит следующие дидактические единицы: управление задачами; управление памятью; управление вводом-выводом; управление файлами; ассемблеры; формальные языки и грамматики; структура компиляторов и интерпретаторов. В соответствии с этим всю дисциплину «Системное программное обеспечение» можно условно разделить на две большие части:

- управление задачами, памятью, вводом-выводом и файлами;
- формальные языки, грамматики и компиляторы.

В учебном пособии предлагается цикл из четырех лабораторных работ, относящихся к первой части дисциплины. При выполнении лабораторных работ студентам сначала предлагается изучить архитектуру памяти процессора и структуру логического диска с помощью системных утилит, а затем написать программы на языке Ассемблера.

Цель практикума заключается в том, чтобы студенты овладели навыками программирования на языке ассемблера для управления клавиатурой, монитором и дисками, работая в разных версиях операционной системы Windows.

В учебном пособии описание лабораторных работ сопровождается необходимыми теоретическими сведениями и контрольными вопросами, что позволяет начать выполнение лабораторных работ одновременно с началом лекционного курса.

1. ОРГАНИЗАЦИЯ ПРАКТИКУМА

1.1. Состав лабораторных работ

Экспериментальные исследования выполняются по мере изучения соответствующего теоретического материала и представляют собой цикл взаимосвязанных лабораторных работ. Практикум состоит из двух частей. В первой изучается управление стандартными символьными устройствами, во второй – управление дисками и файлами.

В первую часть входят две четырехчасовые лабораторные работы:

- изучение логической организации программы на языке Ассемблера и архитектуры памяти процессора, а также знакомство с работой Турбо отладчика на примере готовой программы;
- написание программы на языке Ассемблера для управления клавиатурой и монитором под управлением операционной системы MS DOS по индивидуальному заданию.

Поскольку считается, что операционная система является управляемой по прерываниям, поэтому при написании программы студенты должны усвоить, каким образом организованы программные прерывания, как они вызываются, какие функции выполняют, как обеспечивают обращение к устройствам и когда нужно воспользоваться прерываниями BIOS или MS DOS.

Защита лабораторной работы состоит из демонстрации корректной работы программы. Кроме того, необходимо продемонстрировать покомандное ее выполнение в Турбо отладчике с пояснениями исполнения команд, распределения памяти, установки флагов и организации работы стека.

Во вторую часть практикума входят две лабораторные работы по изучению структуры жесткого диска и структуры логического диска с файловой системой FAT. Во время выполнения первой лабораторной работы студенты после изучения теории организации файловых систем должны написать программы на языке Ассемблера для работы со структурами жестокого диска и логического диска. Поскольку файловая система FAT продолжает достаточно активно использоваться и, кроме того, является самой простой по организации и самой незащищенной из всех современных файловых систем, в ней достаточно легко получить доступ к системным областям любого логического диска. Поэтому именно эта файловая система выбрана для изучения на лабораторных работах по дисциплине «Системное программное обеспечение». Изучение файловой системы FAT можно начать с работы с программой Diskedit, которая позволяет прочитать напрямую системные области дискеты, а также записать в эти области изменения.

После изучения структуры логического диска студентам необходимо написать программу на языке Ассемблера, в которой они должны, выполнить работу по изучению системных областей логического диска, например, найти файл на диске, используя информацию из загрузочного сектора логического диска, области FAT и корневого каталога.

Во второй лабораторной работе решается та же задача организации работы с системными областями дисков, но под управлением одной из версий операционной системы Windows: Windows 95/98/ME или Windows NT/2000/XP/2003/Vista. Поскольку операционная система Windows является многозадачной, то любые операции ввода-вывода объявляются привилегированными и выполняются только самой ОС в режиме ядра. Доступ к устройствам осуществляется через системные вызовы, поэтому при выполнении лабораторной работы надо учитывать эту специфику.

Для студентов, желающих глубже изучить организацию работ файловых систем, рекомендуется начинать работу с чтения главной загрузочной записи жесткого диска, чтобы определить, в каком разделе установлена файловая система FAT. Если на жестком диски нет раздела с этой файловой системой, перейти к работе с дискетой.

Защита лабораторной работы состоит в демонстрации корректной работы программы, написанной в соответствии с заданием, и ответов на вопросы по организации файловых систем.

1.2. Краткие сведения о работе в Турбо отладчике

Турбо отладчик обеспечивает программиста всеми мощными и сложными средствами, сохраняя при этом простоту в использовании. Турбо отладчик можно использовать с любой программой для решения труднейших проблем процесса отладки: поиска места нахождения ошибки и ее причины. Турбо отладчик поможет преодолеть эти трудности с помощью исключительных возможностей по замедлению выполнения программы, благодаря чему можно исследовать состояние программы в любой заданной точке. Можно даже тестировать новые значения переменных, чтобы увидеть, как они воздействуют на программу. Эти возможности реализуются с помощью трассировки, пошагового выполнения, просмотра, изменения и прослеживания.

В данном пособии рассмотрена отладка на уровне Ассемблера и работа с окном CPU.

Турбо отладчик поддерживает полный синтаксис выражений Ассемблера, обеспечивает работу с константами всех типов, которые используются в Ассемблере (byte, word, длинные, составные, с плавающей точкой, вещественные, с двойной и расширенной точностью). Если не используется одно из соглашений Ассемблера по переопределению основания, то целочисленные константы являются шестнадцатеричными.

Турбо отладчик поддерживает большинство операций, использующиеся в Ассемблере. Старшинство этих операций соответствует старшинству операций, принятому в Ассемблере.

С помощью данного окна CPU можно проверять и модифицировать байты данных непосредственно в шестнадцатеричном виде, анализировать стек вызова функций, проверять и модифицировать регистры центрального процессора (ЦП) и его флаги.

Общий вид окна CPU приведен на рис.1.1.

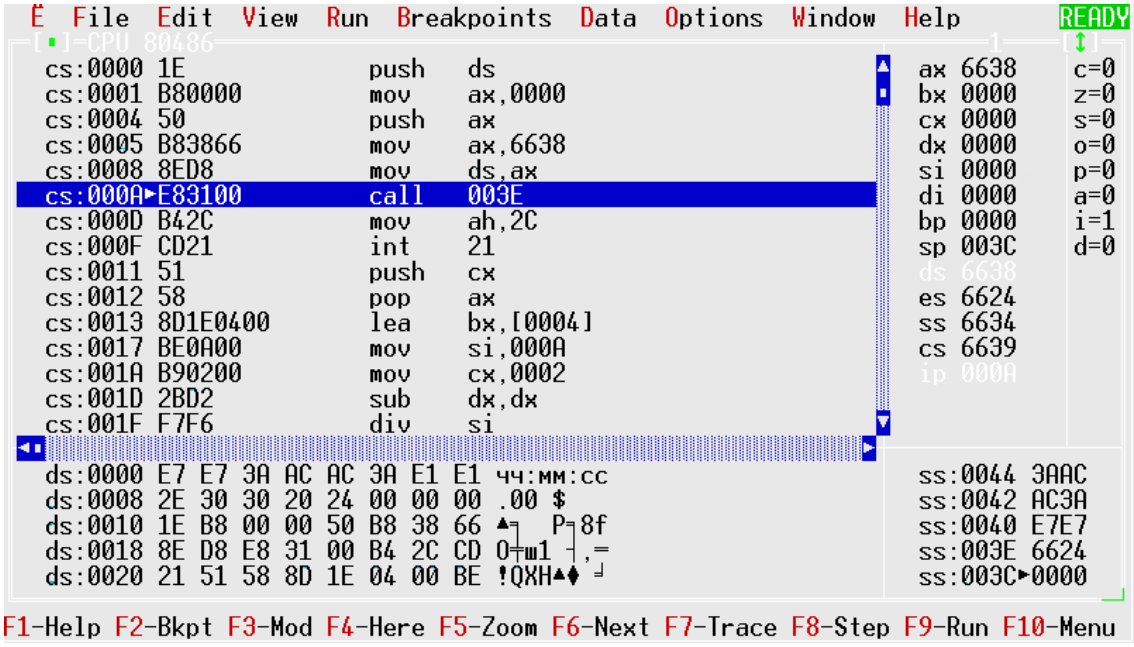

Рис. 1.1. Общий вид окна CPU

В окне CPU (ЦП) показано все состояние центрального процессора. С его помощью можно проверять и изменять биты и байты, составляющие код и данные программы.

Оперативные клавиши:

F1=Help справка;	F6=Next дальше;
F2=Bkpt точка останова;	F7=Trace трассировка;
F3=Close закрыть;	F8=Step шаг;
F4=Here здесь;	F9=Run выполнить;
F5=Zoom переключение окон;	F10=Menu меню.

Окно CPU (ЦП) можно создать, выбрав команду основного меню View/CPU (Обзор/Центральный процессор). В зависимости от того, что просматривается в текущем окне, окно CPU будет позиционировано на соответствующие код, данные или стек. Это предоставляет удобный способ просматривать код, данные или стек (соответствующие текущему положению курсора) "на нижнем уровне". В области кода, данных и стека для смещения начального адреса вывода на 1 байт вверх или вниз можно использовать клавишу Ctrl со стрелками.

В каждой из пяти областей окна CPU (область кода, область данных, область стека, область регистров и область флагов) имеется локальное меню.

Для перехода между областями нужно использовать клавиши Tab или Shift-Tab. Для вывода локального меню в любой области используются клавиши Alt-F10.

В **области кода** по выбранному адресу выводятся дисассемблированные инструкции. Локальная команда *Mixed* (Смесь) позволяет выбрать один из трех способов вывода на экран дисассемблированных инструкций и исходного кода. В окне *Code* (Код) для временной коррекции программы можно использовать встроенный Ассемблер. При этом инструкции вводятся точно так же, как при наборе исходных операторов Ассемблера. Можно также получить доступ к соответствующим данным любой структуры данных, выводя и изменяя их в различных форматах.

В левой части каждой дисассемблированной строки выводится адрес инструкции. Адрес выводится либо в виде шестнадцатеричного значения сегмента и смещения, либо со значением сегмента, замененным именем регистра CS (если значение сегмента совпадает с текущим значением регистра CS). Эта область имеет ширину (которая может переключаться или настраиваться), достаточную для вывода всех образующих инструкцию байт. Дисассемблированная инструкция выводится справа.

После выбора команды *Goto (Переход)* выводится подсказка для ввода нового адреса, на который нужно перейти. Можно ввести адрес, выходящий за пределы программы, что позволяет проверить базовую систему ввода-вывода (BIOS), внутренние области DOS и резидентные утилиты.

Команда *Origin (Начало)* позиционирует вас на текущий адрес программы в соответствии со значениями регистров CS:IP.

Команда *Follow (Следующий)* позиционирует по целевому адресу подсвеченной в данный момент инструкции. Область кода позиционируется заново, чтобы вывести код по адресу, указанному в подсвеченной в этот момент инструкции, по которому будет передано управление. Для условных переходов адрес показывается в случае выполнения перехода. Эту команду можно использовать с инструкциями CALL, JMP, инструкциями условных переходов (JZ, JNE, LOOP, JCXZ и т.д.) и инструкциями INT.

Команда *Caller* (Вызывающая программа) позиционирует на инструкцию, по которой была вызвана текущая подпрограмма или прерывание. Данная команда будет работать не всегда. Если процедура обработки прерывания или подпрограмма занесла в стек элементы данных, то иногда Турбо отладчик не может определить, откуда был выполнен вызов.

Команда *Search* (Поиск) позволяет вводить инструкцию или список байт, которые нужно найти. Следует выполнять поиск только тех инструкций, которые не изменяют байт, в которые они ассемблируются, в зависимости от того, где в памяти они ассемблируются. Например, поиск следующих инструкций проблемы не представляет:

```
PUSH    DX
POP  [DI+4]
ADD  AX,100
```

а попытка поиска следующих инструкций может привести к непредсказуемым результатам:

```
JE    123
CALL     MYFUNC
LOOP     $-10
```

Вместо инструкции можно вводить также список байт.

Команда *Mixed* (Чередование) позволяет выбрать один из трех способов вывода на экран дисассемблированных инструкций и исходного кода:

- *No (Нет)* исходный код не выводится, выводятся только деассемблированные инструкции;
- *Yes (Да)* перед первой дисассемблированной инструкцией, соответствующей данной строке, выводится строка исходного кода. Область устанавливается в данный режим, если исходный модуль написан на языке высокого уровня;
- *Both (Оба)* Для тех строк, которым соответствует исходный код, дисассемблированные строки заменяются строками исходного текста. В противном случае выводятся дисассемблированные инструкции. Используйте этот режим, когда вы отлаживаете модуль на Ассемблере и хотите видеть строку исходного текста, а не соответствующую дисассемблированную инструкцию. Область устанавливается в данный режим вывода, если текущим модулем является исходный модуль Ассемблера.

Команда *New CS:IP* (Новое значение регистров CS:IP) устанавливает значение счетчика адреса программы (регистры CS:IP) в соответствии с текущим (подсвеченным) адресом. При повторном запуске программы выполнение начнется с этого адреса.

Команда *Previous* (Предыдущий) восстанавливает область кода в то состояние (позицию), которое она имела до выполнения команд Goto, Follow или Caller. При использовании команды Previous позиция окна кода запоминается, поэтому повторное использование этой команды приводит к переключению между двумя адресами (туда и обратно). Кроме того, эта команда восстанавливает позицию области кода в соответствии с адресом, который был текущим перед последней командой, явно изменившей его значение.

В **области данных** показано непосредственное содержимое выбранной области памяти. В левой части каждой строки находится адрес данных, выводимых на данной строке. Адрес выводится в виде шестнадцатеричного значения сегмента и смещения или значение сегмента заменяется именем сегмента DS, если значение сегмента совпадает с текущим содержимым регистра DS.

Далее в области выводится непосредственное содержимое элементов данных. Формат этой области зависит от режима вывода, выбранного с помощью команды локального меню Display As (Вывести как ...).

В правой части каждой строки выводятся символы, соответствующие показанным байтам. Турбо отладчик выводит все печатаемые значения, соответствующие байтовым эквивалентам.

Примечание. Если окно данных используется для проверки содержимого дисплейной памяти, данных базовой системы ввода-вывода или векторов в младших адресах памяти, то отображаются значения, находящиеся там во время выполнения отлаживаемой программа. Они не совпадают с теми значениями, которые находятся в указанных областях памяти во время просмотра. Турбо отладчик определяет обращение к областям данных, которые также используются им самим, и извлекает значения этих данных из их копии для программы пользователя.

Команды локального меню области данных приведены в табл. 1.1.

Таблица 1.1

Команды контекстного меню области данных

Команда	Описание
Goto (Переход)	Выводит на экран данные по новому адресу
Search (Поиск)	Выполняет поиск строки или байтов
Next (Следующий)	Выполняет повторный поиск (следующего вхождения)
Change (Изменение)	Изменяет байты данных по адресу курсора
Follow (Следовать)	Следует по цепочке указателя (ближний или дальний тип)
Near Code(Ближний код)	Следует по цепочке указателя (ближний тип). Устанавливает область кода под курсором в ближний адрес
Far Code (Дальний код)	Следует по цепочке указателя (дальний тип).

	Устанавливает область кода под курсором в дальний адрес
Offset to Data(Смещение данных)	Устанавливает область кода в ближний адрес под курсором
Base Segment:0 to Data	Устанавливает область данных в (Базовый сегмент данных) начало сегмента, который содержит адрес под курсором
Previous (Предыдущий)	Выводит на экран данные по последнему адресу
Display As (Режим вывода) - Byte (Байт) - Word (Слово) - Long (Длинный)	На экран выводятся: шестнадцатеричные байты; шестнадцатеричные слова; шестнадцатеричные 32-битовые длинные слова
Comp (Сложный)	На экран выводятся 8-байтовые целые
Float(С плавающей точкой)	На экран выводятся короткие (4-байтовые) числа с плавающей точкой
Real (Вещественный)	На экран выводятся 6-байтовые числа с плавающей точкой

Окончание табл. 1.1

Команда	Описание
Double(С двойной точностью)	На экран выводятся 8-байтовые числа с плавающей точкой.
Extended(С расширенной точностью)	На экран выводятся 10-байтовые числа с плавающей точкой
Block (Блок) Clear (Очистка) Move (Перемещение) Set (Присваивание) Read (Считывание) Write (Запись)	Работа с блоком: очищает блок памяти; перемещает блок в памяти; присваивает блоку памяти значение (побайтно); выполняет чтение из файла в память; записывает из памяти в файл

В **окне Registers (Регистры)** выводится содержимое регистров и флагов ЦП. Данное окно содержит две области, которые эквивалентны областям регистров и флагов окна CPU (ЦП). Значение любого регистра или флага можно изменить с помощью команды локального меню (табл. 1.2).

Таблица 1.2

Команды локального меню окна регистров

Команда	Описание
Increment (Увеличение)	Добавляет 1 к текущему (подсвеченному) регистру
Decrement (Уменьшение)	Вычитает 1 из текущего (подсвеченного) регистра
Zero (Очистка)	Очищает содержимое текущего регистра

Change (Изменение)	Присваивает текущему (подсвеченному) регистру новое значение
Register 32-bit (32-разрядный регистр) No/Yes (Да/Нет)	Переключает экран в режим вывода 32-разрядных регистров

В **окне Stack (Стек)** выводится текущее состояние стека. При этом внизу указаны самые первые вызовы функций, затем – все последующие вызовы в том порядке, как они вызывались.

Можно проверить исходный код любой указанной в стеке функции, переведя на нее подсветку и нажав клавиши Ctrl-I. Подсветив имя функции в стеке и нажав Ctrl-I, открывается окно Variables (Переменные), в котором выводятся переменные, глобальные относительно программы, переменные, локальные относительно функции, и аргументы вызова функции. Команды локального меню окна стека приведены в табл. 1.3.

Таблица 1.3

Команды локального меню окна стека

Goto (Переход)	Выводит на экран содержимое стека по новому адресу
Origin (Начало)	Выводит на экран данные по адресу SS:SP.
Follow (Следовать)	Выводит код, на который указывает текущий элемент
Previous (Предыдущий)	Восстанавливает на экране вывод по последнему адресу
Change (Изменение)	Позволяет редактировать информацию

2. ЯЗЫК АССЕМБЛЕРА ДЛЯ УПРАВЛЕНИЯ КЛАВИАТУРОЙ И МОНИТОРОМ В ОПЕРАЦИОННОЙ СИСТЕМЕ MS DOS

2.1. Постановка задачи

Основной целью выполнения лабораторных работ первой части является:

- знакомство со структурой программы и с работой Турбо отладчика на примере готовой программы, написанной на языке Ассемблера;
- закрепление теоретических знаний по основным приемам программирования на языке Ассемблера и работе со стандартными символьными устройствами с использованием прерываний DOS и BIOS.

Задача. *Используя язык Ассемблера, написать программу для управления клавиатурой и монитором.*

При вводе символьной строки с клавиатуры программа должна выполнять проверку нажатой клавиши таким образом, чтобы обеспечивать не только ввод ASCII-кодов, но и правильную обработку управляющих клавиш, таких, как Backspace, Delete, клавиш управления курсором и т.д. Кроме того, программа должна игнорировать нажатие функциональных клавиш, если их обработка специально не оговорена в задании.

При выводе обработанной строки на экран монитора необходимо сохранить текущий режим работы видеоадаптера, установить требуемый режим, а перед завершением работы программы восстановить сохраненный режим. При переключении режима работы видеоадаптера, если устанавливается нестандартный режим, необходимо проверить, поддерживает ли его видеосистема компьютера.

Прежде чем писать программу на языке Ассемблера необходимо познакомиться с архитектурой процессора, так как Ассемблер является машинно-ориентированным языком и отражает особенности машины, для которой используется.

2.2. Архитектура процессора

При написании ассемблерных программ необходимы знания не только команд Ассемблера, но и организации всей системы компьютера. Язык Ассемблера требует задания действий на уровне внутренних компонентов и в первую очередь на уровне микропроцессора. В данном лабораторном практикуме рассматривается базовый язык Ассемблера для микропроцессора Intel, используемый на персональных ЭВМ.

Процессор делит адресное пространство на произвольное количество сегментов, каждый из которых содержит не более 64 Кбайт. Адрес первого байта сегмента всегда кратен 16 и называется адресом сегмента или параграфом сегмента.

Для работы с памятью используются две шины – шина адреса и шина данных. Физически память устроена таким образом, что возможна адресация как 32-битных и 16-битовых слов, так и отдельных байтов памяти. В любом случае так называемый физический адрес передается из процессора в память по шине

адреса. Ширина шины адреса определяет максимальный объем физической памяти, непосредственно адресуемой процессором.

При работе в реальном режиме процессора компьютер оснащен 20-разрядной шиной адреса и 16-разрядной шиной данных, это означает, что имеется возможность адресоваться к 1 Мбайту памяти. Диапазон физических адресов для 20-разрядной шины можно записать следующим образом:
00000h <= [физический адрес] <= FFFFFh

Однако все регистры процессора являются 16-разрядными, поэтому возникает проблема представления 20-разрядного физического адреса памяти при помощи содержимого 16-разрядных регистров.

Для разрешения этой проблемы используется двухкомпонентный логический адрес. Логический адрес состоит из 16-разрядных компонент: сегмента памяти и смещения внутри сегмента.

Для получения 20-разрядного физического адреса к сегментной компоненте приписываются справа четыре нулевых бита, затем полученное число складывается с компонентой смещения.

Логический адрес принято записывать в форме <сегмент**:** смещение>.

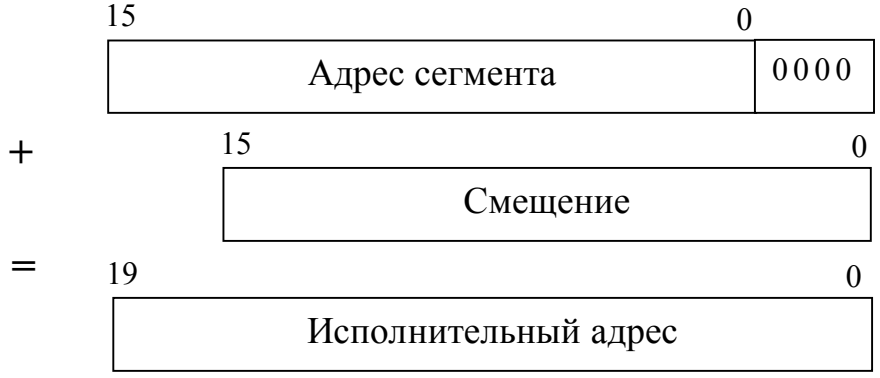

Рис. 2.1. Получение физического адреса

Микропроцессоры имеют 14 специализированных регистров. Четыре сегментных регистра хранят начальные адреса четырех сегментов. Пять регистров – указатели и индексы – хранят смещения, используемые для обращения к сегментам памяти. Четыре арифметических регистра используются для временного хранения промежуточных результатов и операндов арифметических и логических инструкций. Регистр флагов содержит несколько 1-битных флагов, отражающих статус процессора или управляющих режимов исполнения некоторых инструкций. Каждый регистр имеет длину в одно слово (16 бит) и адресуется по имени.

Регистры общего назначения, или арифметические регистры, обозначаются AX, BX, CX и DX (рис.2.2). Каждый из них можно также рассматривать как пару независимо адресуемых восьмиразрядных регистров. Левый байт является старшей частью (high), а правый – младшей частью (low) и соответственно обозначаются старшие – AH, BH, CH и DH, младшие – AL, BL, CL и DL.

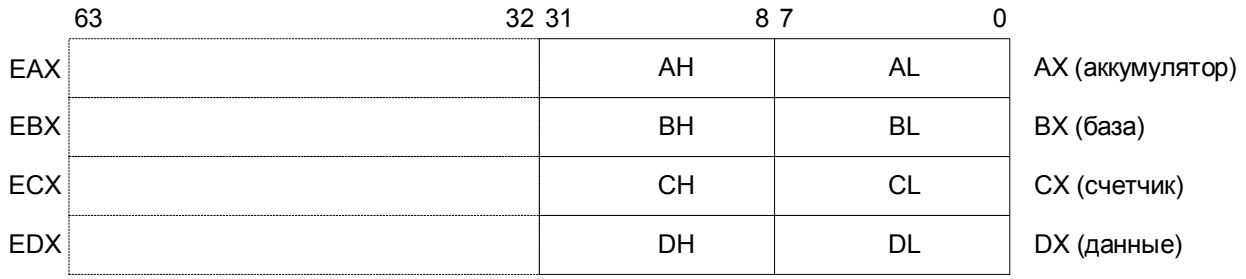

Рис. 2.2. Регистры общего назначения

Каждый из регистров имеет некоторые специальные функции. AX – регистр аккумулятор и основной регистр, используемый в арифметических операциях. Регистр BX часто используется для хранения адреса начала таблицы перекодирования символов, а также может содержать смещение для косвенной адресации. Регистр CX – счетчик числа повторений циклов и блочных пересылок. Регистр DX используется как расширение аккумулятора для операций, дающих 32-разрядный результат. Регистры общего назначения используются также для передачи значений или адресов параметров к подпрограммам.

Каждый сегментный регистр (рис.2.3) обеспечивает адресацию памяти объемом 64 Кбайта, которая называется текущим сегментом. Адрес в сегментном регистре автоматически умножается на 16 для того, чтобы он указывал на одну из 16-байтовых границ мегабайтного адресного пространства микропроцессора.

Регистр CS используется для определения кодового сегмента, содержащего выполняемую программу, и содержит начальный адрес этого сегмента. Адрес в регистре CS складывается со значением смещения в командном указателе (регистр IP) для получения ссылки на текущую команду.

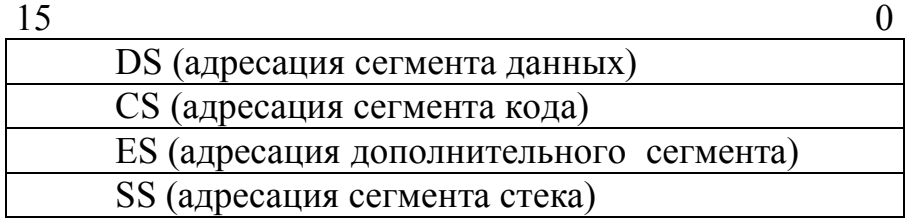

Рис. 2.3. Сегментные регистры

Регистр DS используется для доступа к сегменту данных и содержит его начальный адрес. Этот адрес плюс значение смещения, определенное в команде, указывают на конкретную ячейку в сегменте данных.

Регистр SS задает начальный адрес сегмента стека.

Регистр ES используется, как правило, для адресации данных, находящихся в разных физических сегментах.

Пять регистров смещения (рис.2.4) служат для точного указания адреса байта или слова относительно начала соответствующего сегмента.

15	0
IP (указатель команд)	
SP (указатель стека)	
BP (указатель базы)	
SI (индекс источника)	
DI (индекс назначения)	

Рис. 2.4. Регистры смещения

Регистр IP – указатель команды. Содержит смещение следующей команды относительно начала кодового сегмента. По мере выполнения команд содержимое регистра IP увеличивается на длину команды. Программа не может изменить или получить значение этого регистра. Команды перехода или вызова процедур могут неявно изменить значение IP, сохраняя его значение в стеке или восстанавливая его значение из стека.

Регистры SP и BP обеспечивают системе доступ к данным в сегменте стека. Регистр SP (указатель вершины стека) обеспечивает использование стека в памяти, позволяет временно хранить адреса и данные. Он связан с регистром SS для адресации стека. Регистр BP является указателем базы. Он используется для того, чтобы зафиксировать положение стека в какой-то момент времени и в дальнейшем адресоваться к данным, расположенным в стеке.

Регистры SI (индекс источника) и DI (индекс приемника) используются для формирования сложных адресов, состоящих из смещения начала блока данных в сегменте и относительного смещения элемента данных внутри блока. При этом смещение начала блока обычно хранится в регистре BX или непосредственно в команде, а регистр SI или DI задает смещение внутри блока. Кроме того, индексные регистры участвуют в выполнении команд, работающих со строками байтов или слов.

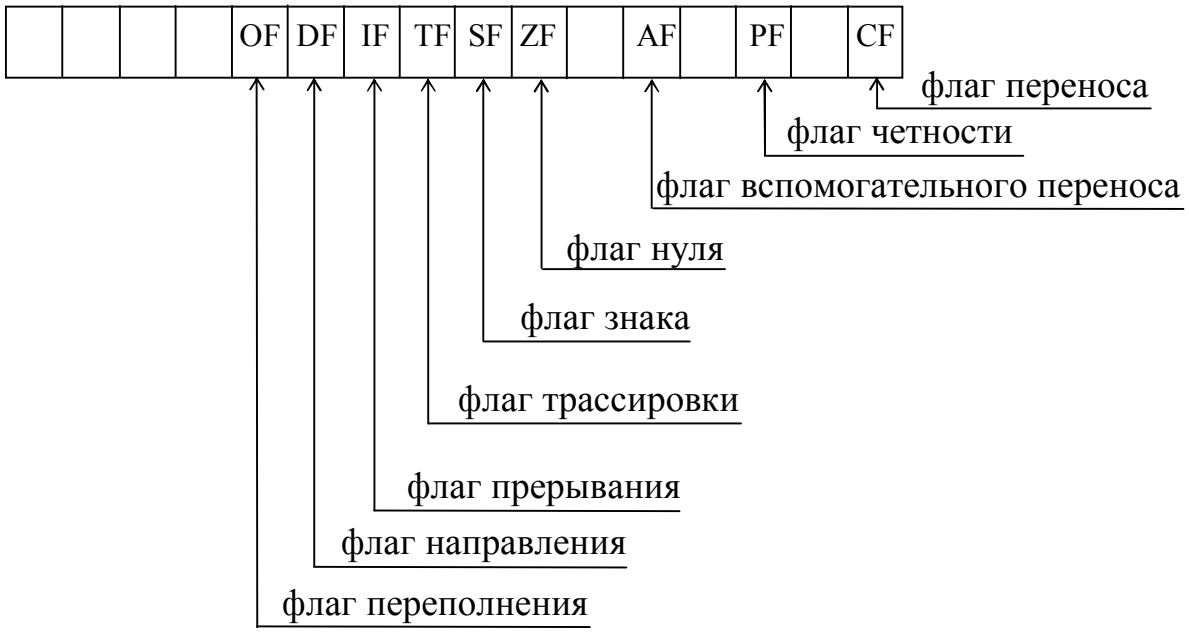

Рис. 2.5. Регистр флагов

Регистр флагов представляет собой набор отдельных статусных и управляющих битов, называемых флагами. Флаги объединены в 16-разрядный регистр, но чаще всего используются независимо друг от друга. Девять из 16 бит флагового регистра являются активными и определяют текущее состояние машины и результаты выполнения. Все флаги младшего байта устанавливаются арифметическими или логическими операциями микропроцессора. Все флаги старшего байта, за исключением флага переполнения OF, устанавливаются и сбрасываются специально предназначенными для этого командами.

Неиспользуемые биты регистра флагов установлены в 0.

2.3. Стек

Стек обеспечивает программам место для хранения и слежения за процессом работы. Наиболее важное использование стека – это запись адреса программы, откуда была вызвана подпрограмма, и параметров, передаваемых подпрограммам.

Размер стека, как и любого сегмента, ограничен 64 Кбайтами. Стек используется от основания, к вершине, т.е. с наибольшего адреса к наименьшему адресу (рис. 2.6). Запись данных в стек – операция загрузки. При загрузке нового элемента указатель вершины стека уменьшается на 2, т.е. вершина стека приближается к базовому адресу. Съем данных из стека начинается с вершины стека и называется операцией считывания. При этом значение указателя вершины

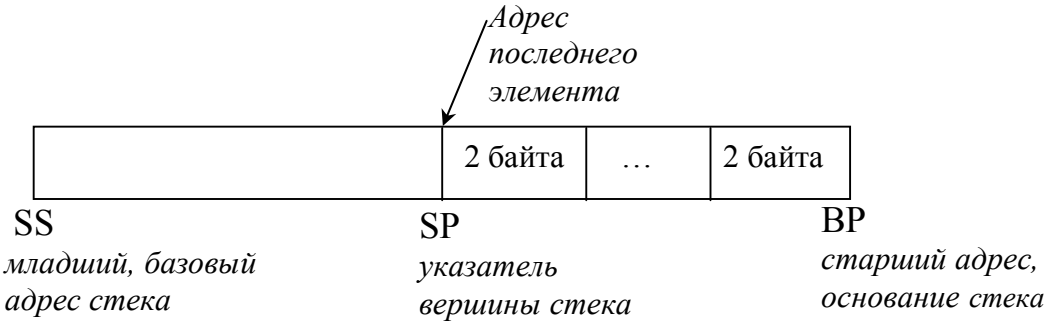

Рис. 2.6. Стек программы

стека увеличивается на 2.

Микропроцессор изменяет содержимое регистра SP в соответствии с работой стека, а содержимое регистра SS не изменяется никакими стековыми операциями. Система загружает регистр SS, и он определяет сегмент, в котором располагается стек.

Любая часть программы может создавать новую область стека. Но обычно при работе программы создается один стек, который используется любыми обслуживающими подпрограммами, вызываемыми в процессе исполнения программы. Когда программа не работает, DOS использует свой собственный стек.

Трудно определить размер стека, который потребуется программе. Кроме того, в операционной системе не предусмотрено автоматическое средство для определения выхода за границу стека. Это заставляет программиста самого заботиться о том, какое пространство отвести под стек. Если отвести под стек памяти меньше, чем требуется для запоминания адресов, предыдущее содержимое стека стирается и можно «запортить» какую-то часть программы или данных или выйти за пределы физической памяти.

2.4. Система прерываний в реальном режиме работы процессора

Семейство компьютеров IBM в значительной степени управляется с помощью прерываний, генерируемых аппаратным и программным обеспечением, т.е. в реальном режиме имеется два типа прерываний: аппаратные и программные. Программные прерывания инициируются командой INT, аппаратные – внешними событиями, асинхронными по отношению к выполняемой программе. Обычно аппаратные прерывания инициируются аппаратурой ввода-вывода после завершения выполнения текущей операции. Кроме того, некоторые прерывания зарезервированы для использования самим процессором.

Для обработки прерываний в реальном режиме процессор использует таблицу векторов прерываний. Сегментные адреса, используемые для определения местоположения программ обработки прерываний, называются векторами прерываний. Таблица векторов прерываний располагается в самом начале оперативной памяти, т.е. ее физический адрес – 00000.

Таблица векторов прерываний состоит из 256 элементов по 4 байта, поэтому ее размер составляет 1 Кбайт. Вектора состоят из 16-битового сегментного адреса процедуры обработки прерываний и 16-битового смещения. Смещение хранится по младшему адресу, сегментный адрес – по старшему. Каждый вектор прерывания имеет свой номер, называемый номером прерывания, который указывает на его место в таблице. Этот номер, умноженный на 4, дает абсолютный адрес первого байта вектора в рабочей памяти.

Каждый вектор прерывания получает свои значения при запуске системы, но позже пользователь может их изменить. Сначала управление передается к BIOS. BIOS выполняет серию тестов и процедур инициализации, задавая значения определенных векторов прерываний. После этого BIOS выполняет процедуру начальной загрузки и передает управление DOS, которая также задает значения определенным векторам. Кроме того, DOS переназначает некоторые из векторов BIOS к своим подпрограммам.

Когда происходит программное или аппаратное прерывание, текущее состояние регистров CS, IP, а также регистра флагов записываются в стек программы. Далее из таблицы векторов прерываний выбираются новые значения для CS и IP, при этом управление передается на процедуру обработки прерываний.

Перед входом в процедуру обработки прерываний принудительно сбрасываются флаги трассировки TF и разрешения прерываний IP.

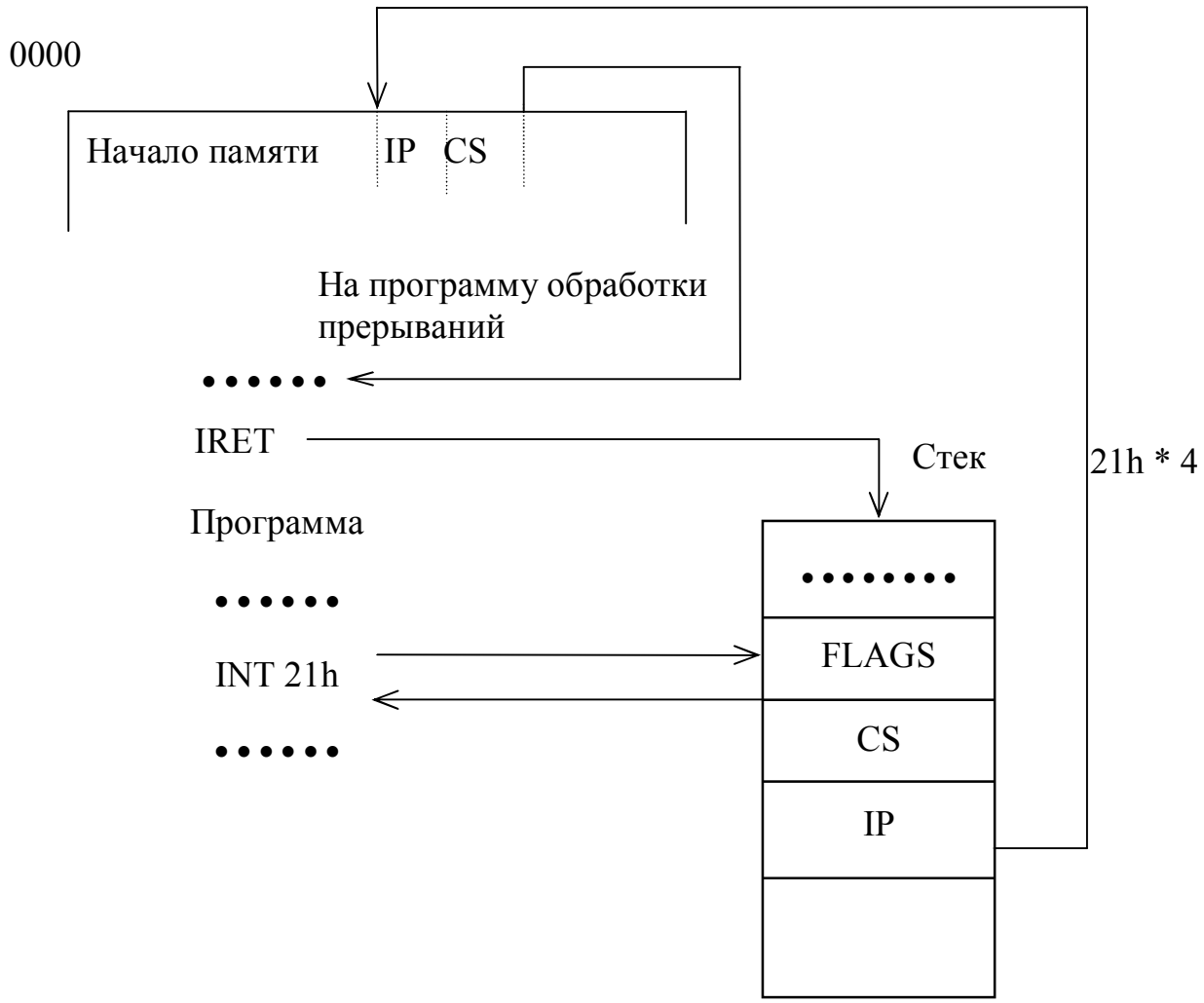

Рис. 2.7. Схема обработки прерывания

Завершив обработку прерывания, процедура должна выдать команду IRET, по ней из стека будут извлечены значения для регистров CS, IP и регистра флагов. Далее продолжается выполнение прерванной программы. Схема обработки прерывания показана на рис.2.7.

2.5. Структура программного модуля на языке Ассемблера

Структура программы на языке Ассемблера соответствует организации памяти микропроцессора. Отдельный программный модуль может содержать от одного до четырех сегментов:

- кодовый сегмент;
- сегмент данных;
- расширенный сегмент данных;
- стековый сегмент.

Кодовый сегмент содержит, как правило, команды, подлежащие выполнению. Этот сегмент в свою очередь может разбиваться на части директивами определения процедур.

Сегмент данных содержит переменные, константы, таблицы и другие данные программного модуля. В языке Ассемблера отсутствует понятие типов

данных, для них определяется только размер. За корректным использованием данных должен следить программист.

Расширенный сегмент данных – это дополнительный информационный сегмент программы. Этот сегмент используется, когда нужно обратиться к каким-либо областям памяти, находящимся вне приложения, например прочитать информацию в клавиатурном буфере или обратиться напрямую к видеопамяти.

Стековый сегмент – это область памяти, которая используется для временного хранения промежуточных данных, необходимых для информационной связи программ или процедур.

Программный модуль на языке Ассемблера должен заканчиваться директивой END. В этой директиве можно указать адрес, с которого начинается выполнение программы.

Все программы, написанные на языке Ассемблера, имеют схожую структуру:

```
.386                    ;это ассемблерная директива, говорящая
                        ;ассемблеру использовать набор операций для
                        ;процессора 80386.
.model small            ;это ассемблерная директива, определяющая
                        ;модель памяти программы.
.DATA                   ;объявление сегмента данных
<Инициализируемые данные>
.....
.CODE                   ;объявление кодового сегмента
<метка>
<Код>
.....
end <метка>             ;конец программы
```

Объявление сегментов позволяет Ассемблеру проследить, какие из сегментов доступны через сегментные регистры.

Следует обратить внимание на то, что обязательно нужно загрузить адрес начала сегмента данных в регистр DS, так как операционная система этого не делает.

В программе на рис. 2.9 описаны:

- сегмент данных (строки 2 – 3), содержащий описание сообщения для вывода на экран;
- сегмент стека (строка 4) размером 256h байтов;
- кодовый сегмент (строки 5 – 16), в котором выводится сообщение на экран (строки 9 – 11), вводится символ без отображения на экране (строки 12 – 13) и завершается работа программы (строки 14 – 15). В строках 7 и 8 в регистр DS заносится адрес сегмента данных, так как при загрузке программы в оперативную память система не загружает регистр DS.

Команда	Комментарии	№ строки
.model small	;модель памяти	1
.data	;сегмент данных	2
message db 'Symbol: $'	;выводимое приглашение для ввода ;символа	3
.stack 256h	;размер стека 256h байт	4
.code	;описание сегмента кода	5
main:	;начало основной программы	6
mov ax,@data	;загрузка адреса	7
mov ds,ax	;сегмента данных	8
lea dx,message	;загрузка эффективного адреса ;сообщения в регистр DX	9
mov ah,09h	;функция вывода строки, ;завершающейся знаком $, на экран ;(регистр DX содержит адрес ;сообщения)	10
int 21h	;прерывание DOS	11
mov ah,0	;считываем символ без эха	12
int 16h	;прерывание BIOS	13
mov ah,4ch	;функция для выхода в ОС	14
int 21h	;прерывание DOS	15
end main	;конец основной программы	16

Рис. 2.8. Пример программного модуля на языке Ассемблера

2.6. Ввод данных с клавиатуры

Работой клавиатуры управляет специальная электронная схема – контроллер клавиатуры. В его функции входит распознавание нажатой клавиши и помещение закрепленного за ней кода в свой выходной регистр (порт), обычно с номером 60h. Код клавиши, поступающий в порт, называется скэн-кодом и является, по существу, порядковым номером клавиши. При этом клавише присвоены два скэн-кода, отличающиеся друг от друга на 80h. Один скэн-код засылается контроллером в порт 60h при нажатии клавиши, другой при ее отпускании. Нажатие и отпускание любой клавиши вызывает сигнал аппаратного прерывания INT 09h.

При нажатии клавиши программа INT 09h считывает из порта 60h ее скэн-код нажатия и по таблице трансляции скэн-кодов в коды ASCII формирует двухбайтовый код, старший байт которого содержит скэн-код, а младший – код ASCII. При этом скэн-код характеризует клавишу, а код ASCII определяет закрепленный за ней символ. Полученный двухбайтовый код засылается программой INT 09h в кольцевой буфер ввода, который служит для синхронизации процессов ввода данных с клавиатуры и приема их выполняемой программой.

Объем кольцевого буфера составляет 15 слов. Символы извлекаются из него в том же порядке, в котором поступили. За состоянием буфера следят два указателя. В хвостовом указателе (40:1Ch) хранится адрес первой свободной ячейки, в головном указателе (40:1Ah) – адрес самого старого кода, принятого с клавиатуры и еще не востребованного программой. Если оба указателя равны – буфер пуст. Программа Int 09h, сформировав двухбайтовый код, помещает его по адресу, находящемуся в хвостовом указателе, после чего этот адрес увеличивается на 2, опять указывая на первую свободную ячейку.

Кроме кодов ASCII существуют расширенные коды, присвоенные клавишам или комбинациям клавиш, которые не имеют представляющего их символа ASCII, таким, как функциональные клавиши или комбинации с клавишей Alt. Расширенные коды имеют длину 2 байта, причем первый всегда ASCII 0. Второй байт – номер расширенного кода.

Операционная система предоставляет несколько способов ввода данных с клавиатуры:

– обращение к клавиатуре как к файлу, с помощью прерывания DOS INT 21h с функцией 3Fh;

– использование группы функций DOS INT 21h из диапазона 1h...Ch, обеспечивающих посимвольный ввод с клавиатуры в разных режимах;

– посимвольный ввод путем обращения непосредственно к драйверу BIOS с помощью прерывания INT 16h.

2.7. Вывод информации на экран

Видеосистема компьютера включает в себя ряд аппаратных и программных средств, позволяющих получать на экране терминала текстовые и графические изображения.

Программные средства обслуживания экрана включают в себя видеоадаптер BIOS, к которому можно обратиться из прикладной программы с помощью прерывания Int 10h и который обеспечивает нижний уровень управления (вывод символов, работа с курсором, переключение режимов видеоадаптера и т.д.), а также программы DOS, активизируемые с помощью прерывания Int 21h и предоставляющие более высокий уровень сервиса в текстовом режиме.

Работа в графическом режиме не обслуживается DOS и может осуществляться только с помощью функций видеодрайвера BIOS.

Существует несколько стандартных режимов работы видеоадаптеров, определенных фирмой IBM. Любой из этих режимов можно инициировать конструкцией типа:

```
mov ah,00h
mov al,Mode          ;Номер видеорежима
int 10h
```

В текстовом режиме изображение состоит обычно из 25 строк по 80 символов в строке. Поскольку таблицы, описывающие форму символов, загружаются в память программно, имеется возможность работать с символами любой конфигурации. Обычно используется стандартная кодовая таблица символов, содержащая знаки английского и русского алфавитов, цифры, специальные символы, символы псевдографики и др.

В графических режимах произвольное изображение рисуется пикселями. Цвет пикселей на экране, как и цвет фона, задаются содержимым цветовых регистров, определяющих цветовую палитру.

Текстовую информацию в графических режимах можно отобразить, используя любые функции DOS и BIOS.

Чтобы сократить время на вывод изображения, можно работать не с функциями BIOS, а непосредственно с видеопамятью.

Для адаптеров типа SVGA был разработан единый стандарт для видеокарт, получивший название VESA (Video Electronics Standards Association). Этот стандарт определен как дополнение к VideoBIOS. Некоторые разработчики прошивают VESA непосредственно в Video ROM BIOS, другие же поставляют VESA в виде загружаемого драйвера.

Для инициализации VESA режима необходимо использовать следующую конструкцию:

```
mov ax,4F02h
mov bx,VESA_mode          ; Номер VESA режима
int 10h
```

Перед вызовом функции рекомендуется проверить, поддерживает ли адаптер этот режим:

```
mov  ax,4F01h
```

```
mov   cx,VESA_mode    ; номер VESA  режима
mov   di,buffer
int   10h
```

Пара регистров ES:DI содержит указатель на буфер размером 256 байт для таблицы описания режима, где в случае успешного завершения находятся атрибуты режима, атрибуты окон A и B и другая информация.

Операционная система предоставляет несколько способов вывода текстовой информации на экран:

– обращение к экрану как к файлу с помощью прерывания DOS INT 21h с функцией 40h;

- использование группы функций DOS INT 21h из диапазона 1h...Ch, обеспечивающих посимвольный вывод и вывод строк;

– вывод путем обращения непосредственно к драйверу BIOS с помощью прерывания INT 10h.

Работа в графическом режиме не поддерживается DOS и может осуществляться только с помощью функций видеодрайвера BIOS.

2.8. Примеры программ

Сначала можно рассмотреть выполнение программы, приведенной на рис. 1 в Турбо отладчике.

При загрузке программы в оперативную память все символические метки и имена заменяются конкретными адресами. Турбо отладчик деассемблирует загрузочный модуль, при этом окно CPU имеет вид, представленный на рис. 2.9. Покомандное выполнение программы осуществляется с помощью функциональных клавиш F7 (с заходом в процедуры) и F8 (без захода в процедуры).

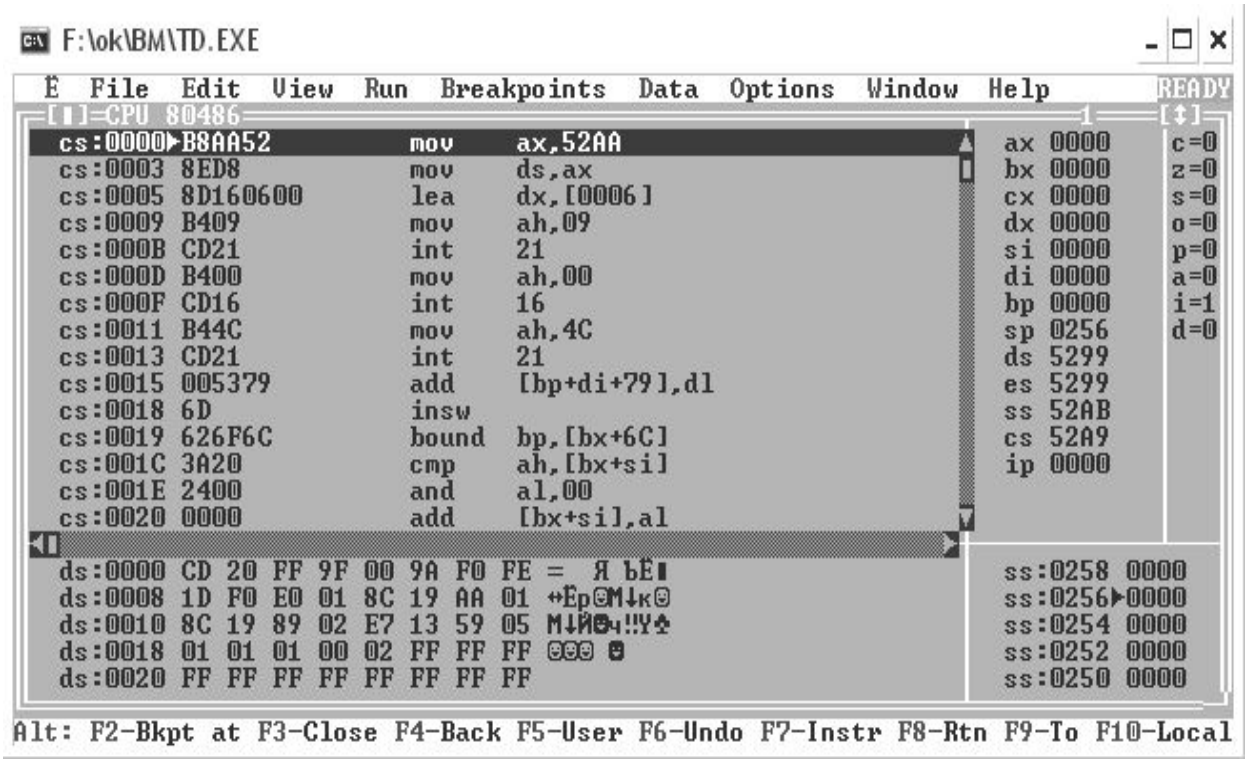

Рис. 2.9. Вид окна CPU при загрузке программы

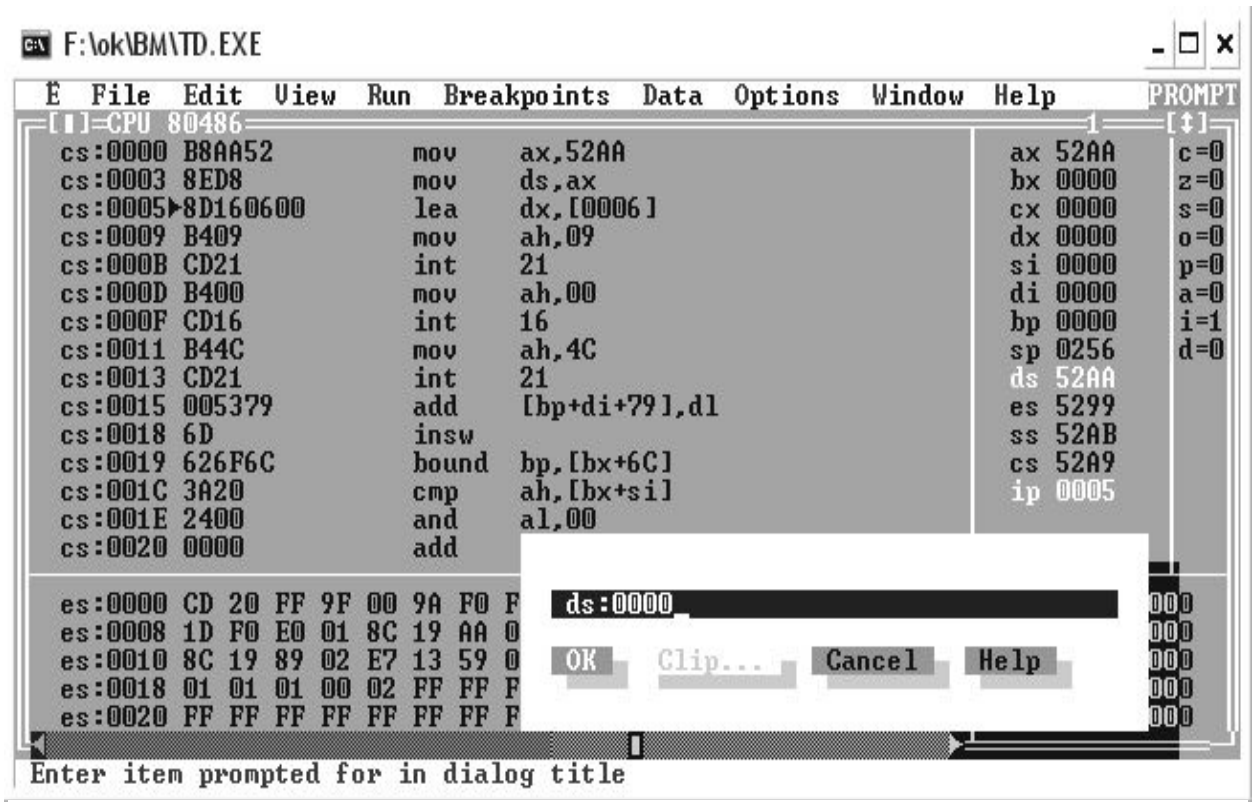

Рис. 2.10. Настройка сегмента данных

Пока не загружен регистр DS адресом сегмента данных программы, содержимое этого сегмента увидеть сложно. Состояние областей окна CPU после выполнения двух первых команд и настройки области данных на сегмент программы (рис. 2.10) показано на рис. 2.11.

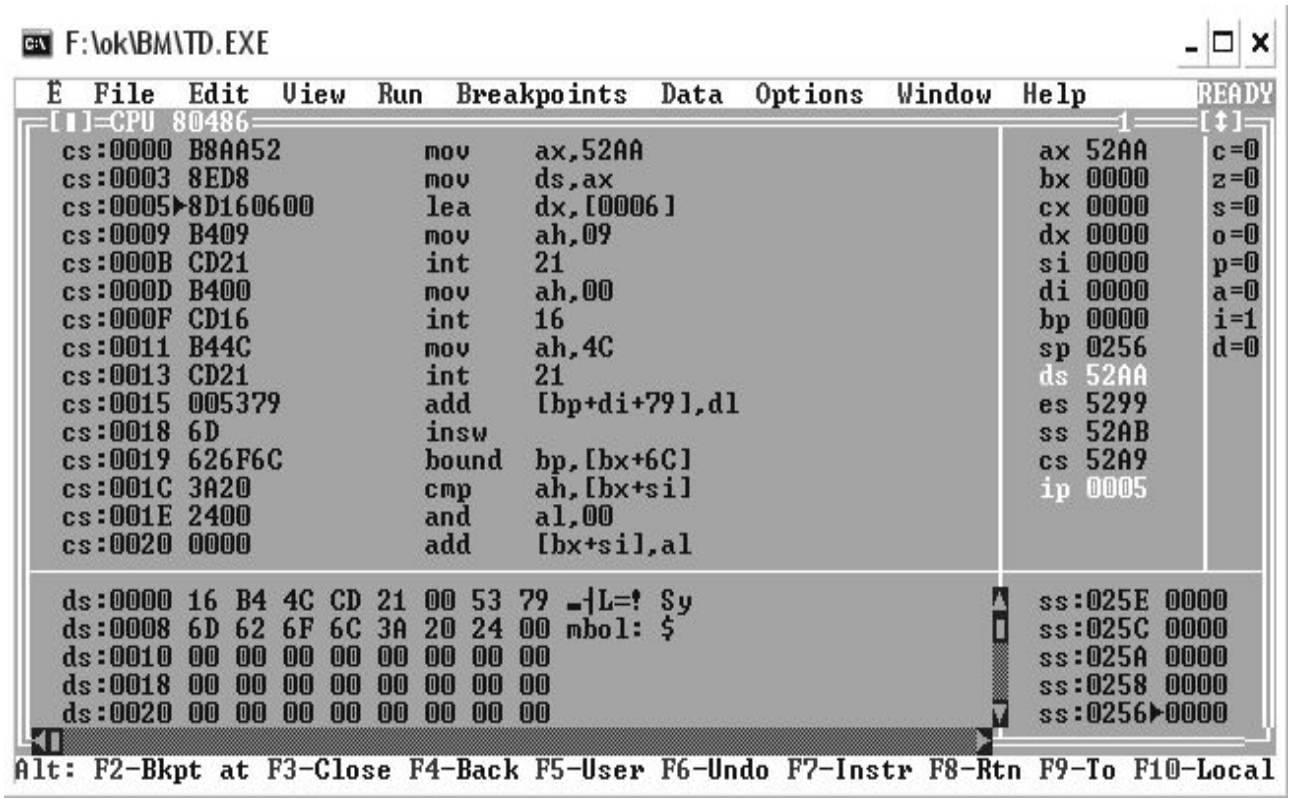

Рис. 2.11. Выполнение программы

При выполнении очередной команды состояние областей окна CPU изменяется. Всегда изменяется содержимое регистра IP, поскольку он является счетчиком команд и при выполнении команды его содержимое увеличивается на длину выполненной команды.

Если при выполнении команды изменяется содержимое какого-либо регистра или флага, то они выделяются белым цветом. На рис. 2.11 видно, что после выполнения команды загрузки регистра DS изменилось содержимое регистров DS и IP. При выполнении следующей команды с адресом CS:0005 в регистр DX загружается смещение строки *message* для того, чтобы вывести эту строку на экран функцией 09 (команда с адресом CS:0009 загружает номер функции в регистр AH) прерывания 21h (вызов прерывания – команда с адресом CS:000B). Следующие две команды используются для ввода символа с клавиатуры. И последние две команды завершают работы программы. При выполнении программы регистры CS:IP содержат логический адрес команды, которая будет выполнена следующей. Для вычисления физического адреса команды содержимое регистра CS умножается на 16 (или 10h) и складывается с содержимым регистра IP. Например, на рис. 2.11 следующей выполняемой командой будет команда

lea dx,[0006]

Ее физический адрес равен 52A9h * 10h + 0005h = 52A95h.

Еще один пример программы с подробными комментариями представлен ниже.

Задача. Ввести строку символов и отдельный символ. Сформировать новую строку, в которой введенный символ вставляется после каждого символа исходной строки.

```
.model      small                   ;модель памяти
.data                               ;сегмент данных
message1    db    'Symbol: $'       ;выводимое приглашение для ввода ;символа
message2    db    0ah,0dh,'Stroka: $' ;выводимое приглашение для ввода ;строки
message3    db    0ah,0dh,'Result: $' ;выводимая строка-результат

FullData STRUC                      ;описание структуры данных для ;хранения
                                    символов
        one   db 9 dup(36)
        two   db 10 dup(36)
        three db 1 dup(36),36
FullData ENDS

        FD FullData1 dup(<>)        ;выделяем память для структуры
.stack 256h                         ;определяем размер стека в 256h ;байт
.code                               ;описание сегмента кода

INPUT       proc  near              ;процедура для ввода символов
        Xor   di,di                 ;очищаем DI-счетчик символов
m1:                                 ;метка m1
        mov   ah,0                  ;считываем символ без эха
        Int   16h                   ;функцией BIOS
        cmp   al,0                  ;проверка нажатой клавиши
        Jne   m3                    ;если она нефункциональная, идем ;на m3
        mov   ah,0                  ;в противном случае вызываем ;снова эту же
        int   16h                   функцию BIOS
        jmp   m1                    ;переходим на начало (m1)
m3:                                 ;метка m3
        cmp   dx,1                  ;проверяем на возможность обрыва ;строки
                                    (dx заполняется до ;вызова функции)
        Je    m4                    ;если флаг установлен, переход ;на m4
        cmp   al,13                 ;если нажата клавиша Enter, то
        Je    quit                  ;переход на конец
m4:                                 ;метка m4
        cmp   al,33                 ;проверяем, не является ли ;введенный
                                    символ управляющим
        Jle   m1                    ;если он все-таки управляющий, ;переход на
                                    m1 (нам не нужны ;такие символы)
        mov   bx[di],al             ;в противном случае записываем в ;буфер
                                    ввода (описанную ;структуру) введенный
```

```
                                символ
        Inc     di              ;увеличиваем счетчик
        mov     ah,2            ;подготавливаем функцию для
        mov     dl,al           ;вывода на экран
        Int     21h             ;прерывание DOS
        loop    m1              ;все это выполняется в цикле
quit:                           ;метка quit (выход)
        Ret                     ;возврат из подпрограммы
INPUT endp                      ;конец описания процедуры

MESSAGE proc   near             ;процедура для вывода сообщений
        push    ax              ;сохраняем значение регистров AX
        push    cx              ;и CX
        mov     ah,09h          ;вызываем функцию вывода строки,
                                ;завершающейся знаком $ на экран  ;регистр
                                dx заполняется до ;вызова процедуры)
        Int     21h             ;прерывание DOS
        pop     cx              ;восстанавливаем значения ;регистров CX
        pop     ax              ;и AX
        Ret                     ;возврат из подпрограммы
MESSAGE endp                    ;конец описания процедуры

main:                           ;основная программа
        mov     ax,@data        ;загрузка адреса
        mov     ds,ax           ;сегмента данных
        Lea     dx,message1     ;загрузка эффективного адреса ;сообщения в
                                DX
        call    MESSAGE         ;выводим сообщение
        Lea     bx,FD.three     ;подготавливаем место, на ;которое будет
                                записан символ ;для вставки
        mov     cx,1            ;количество символов – 1
        mov     dx,1            ;флаг, говорящий о том, что ;ENTER не
                                будет учитываться
        call    INPUT           ;вызов функции ввода символа

        Lea     dx,message2     ;загрузка эффективного адреса ;сообщения в
                                DX
        call    MESSAGE         ;выводим сообщение

        Lea     bx,FD.two       ;подготавливаем место, на ;которое будет
                                записана строка ;символов
        mov     cx,10           ;максимальный размер – 10
        Xor     dx,dx           ;ENTER будет учитываться
        call    INPUT           ;вызов функции ввода символов
        mov     cx,di           ;записываем в CX количество ;фактически
```

```
                                    введенных символов
 mov ch, 36                         ;записываем символ для ;заполнения
                                    освободившегося ;места в строке
        mov   ah,FD.three           ;записываем символ для замены из
                                    ;структуры
        Xor   si,si                 ;чистим счетчики SI
        Xor   di,di                 ;и DI
here:                               ;метка here (начало цикла ;вставки)
        mov   al,offset FD.two[si]  ;записываем в AL введенные ;символы по
                                    порядку из строки
        mov   offset FD.two[si],ch  ;заменяем освободившийся символ ;(он уже
                                    записан в AL) на ;значение для замены (он
                                    ;находится в CH)
        mov   offset FD.one[di],ax  ;записываем символы из регистра ;AX
                                    ;(в младшем байте – символ из ;строки,
                                    старшем – символ для ;вставки)
        add   di,2                  ;увеличиваем счетчик символов в ;новой
                                    строке (со вставленным ;символом) на 2
        Inc   si                    ;счетчик считанных из строки ;символов
                                    увеличиваем на 1
        loop  here                  ;все это выполняем в цикле
        Lea   dx,message3           ;загрузка эффективного адреса ;сообщения
                                    ;в регистр DX
        call  MESSAGE               ;выводим сообщение
        Lea   dx,FD                 ;загрузка эффективного адреса ;сообщения
                                    ;в регистр DX
        call  MESSAGE               ;вывод результирующей строки

        mov   ax,4c00h              ;функция для выхода в ОС
        Int   21h                   ;функция DOS
end     main                        ;конец основной программы
```

Результат выполнения программы представлен на рис. 2.12.

Рис 2.12. Результат работы программы

Вопросы для самопроверки

1. Обоснование необходимости изучения языка Ассемблера.
2. Логический адрес. Его компоненты.
3. Получение физического адреса из логического.
4. Принцип работы стека.
5. Типы прерываний.
6. Вектор прерывания.
7. Структура программного модуля на языке Ассемблера.
8. Варианты использования дополнительного сегмента данных.
9. Необходимость загрузки адреса сегмента данных в регистр DS в начале работы программы.
10. Какой из сегментов является обязательным?
11. Скэн-код клавиши.
12. ASCII-код клавиши.
13. Расширенные коды клавиш.
14. Принцип работы клавиатурного буфера.
15. Байты статуса.
16. Обработка прерывания от клавиатуры.
17. Стандартные режимы работы видеоадаптеров.
18. Характеристики графических режимов работы видеоадаптеров.
19. Характеристики и особенности VESA-режимов.
20. Прямое кодирование цвета в видеоадаптерах SVGA.

Примерные задания для выполнения лабораторной работы

1	Ввести три строки. Первый символ «0», остальные – произвольные. Начиная со второго символа, заменить все «0» на символ, введенный с клавиатуры. Результат – на экран
2	Ввести строку произвольной длины. Конец ввода – ENTER. Полученную строку рассортировать по возрастанию кодов. Вывести, используя атрибуты цвета, в разные части экрана
3	Ввести строку произвольной длины. Конец ввода – ENTER. Полученную строку рассортировать по убыванию кодов
4	Ввести строку произвольной длины. Конец ввода – ENTER. Полученную строку рассортировать по убыванию или возрастанию в зависимости от запроса
5	Ввести символ и подсчитать количество единичных битов в двоичном представлении символа
6	Ввести два символа, подсчитать количество ненулевых байтов в символе и вывести символ с наибольшим количеством ненулевых битов
7	Ввести строку символов и вывести на экран двоичное представление третьего символа
8	Ввести номера строки и столбца и поместить туда курсор. Вывести сообщение о местоположении курсора
9	Ввести два символа. Первую видеостраницу заполнить первым символом – красное на синем. Вторую страницу заполнить вторым символом – синее на красном
10	Ввести строку символов произвольной длины. Конец ввода – ENTER. 1-й, 3-й, 5-й и т.д. символы вывести в 1-ю строку экрана; 2-й, 4-й, 6-й и т.д. – в 5-ю строку экрана
11	Ввести две строки символов. Сформировать три новые строки: в 1-й строке только цифры; во 2-й строке – буквы; в 3-й – специальные символы
12	Ввести три строки символов и удалить из этих строк символ, введенный с клавиатуры. После удаления символа строки сжать
13	Ввести три строки и отдельный символ. Проверить строки на наличие этого символа. Если такого символа в строке нет, поместить его в начало
14	Из неупорядоченного списка удалить min и max элементы.
15	Ввести несколько строк. Сосчитать количество строк и количество символов в каждой строке
16	Организовать изменение цвета фона и текста в диалоге
17	Организовать движение шара по экрану со сменой цвета шара после касания края экрана
18	Ввести символы. При выводе на экран создать эффект «мерцающих букв»

19	Ввести три строки символов и вывести символы каждой строки в обратном порядке
20	Ввести троку символов и сдвигать ее влево и вправо с помощью клавиш управления ← →. При движении изменяется цвет строки
21	Ввести строку символов и сдвигать ее вверх и вниз по экрану, используя клавиши управления ↓ ↑
22	Ввести строку символов и сдвигать ее по экрану вверх, вниз, вправо и влево, используя клавиши ← ↓ ↑ →
23	Изменить форму курсора, ввести строку символов, вывести ее в обратном порядке. Восстановить курсор
24	Ввести символы. При выводе на экран создать эффект «падающих букв»
25	Ввести строку и вывести ее в алфавитном порядке при нажатии клавиши «*»
26	Ввести две строки символов. Посчитать количество символов в каждой строке и вывести еще одну строку, количество символов в которой рано сумме символов двух первых строках
27	Создать 2 страницы и сдвигать 0-ю вправо, 1-ю влево с помощью клавиш ← →. Освободившееся место заполнять пробелами, используя другой цвет Ввести символы и заполнить ими 2 страницы видеопамяти
28	Заполнить 2 видеостраницы символами и сдвигать 0-ю страницу вверх, 1-ю вниз с помощью клавиш ↓ ↑. Освободившееся место заполнять «*»
29	Ввести строку символов и заполнить ими 2 видеостраницы. Сдвигать 0-ю страницу вправо и вверх, 1-ю влево и вниз
30	Создать «звездное небо», «звезда» – символ, введенный с клавиатуры
31	Ввести строку символов. Конец ввода – ENTER. Установить графический (стандартный) режим. Ввести текст на экран в графическом режиме. Изменять цвет фона
32	Ввести строку символов. Вывести строку в разных частях экрана разным цветом (текстовый режим)
33	Ввести строку символов. Организовать в центре экрана окно и вывести туда строку в обратном порядке
34	Отобразить на экране работу клавиатурного буфера
35	Запустить программу с паролем. Если пароль вводится неправильно три раза – завершить программу. При каждом вводе пароля выводить на экран сообщение правильно или неправильно введен пароль
36	Организовать циклическую работу программы. Выход из цикла при нажатии любой клавиши. При выходе из программы вывести на экран сообщение «Работа программы закончена»

37	Ввести строку символов. Сравнить ее с заданной строкой. Вывести на экран в режиме телетайпа сообщение о результате сравнения строк, используя атрибуты цвета
38	Ввести строку символов. Сформировать новую строку, в которой символы разделены пробелами. Вывести обе строки на экран в режиме телетайпа с использованием атрибутов цвета
39	Ввести строку символов произвольной длины. Удалить из строки 2-й и 4-й символ. Вывести обе строки на экран в окно, расположенное в правом верхнем углу. Использовать атрибуты цвета
40	Ввести строку символов, состоящую из букв и цифр. Найти максимальную цифру и ее порядковый номер в строке. При выводе использовать атрибуты цвета
41	Одинаковые ASCII коды имеют клавиши Backspace и Ctrl+H Tab и Ctrl+I Enter и Ctrl+M Escape и Ctrl+[Определить какие клавиши нажаты и вывести сообщение на экран. При выводе использовать атрибуты цвета
42	Создать меню из 3 пунктов. Выбор пункта меню клавишами F1, F2, F3. При нажатии клавиши выдать сообщение на экран и вернуться в меню. При выводе использовать атрибуты цвета. Нажатие остальных клавиш игнорируется
43	Вывести на экран текущую дату и текущее время в форматах дд.мм.гг и чч.мм.ss. Запросить пароль и ввести его равным kss или iмм, где ss – секунды, мм – минуты. Вывести сообщение на экран о правильности введенного пароля
44	Ввести последовательность чисел. Сложить max и min числа и вывести результат на экран
45	Ввести последовательность чисел. Вывести на экран четные числа
46	Ввести строку символов. Проверить, сколько введено парных элементов. Результат вывести на экран
47	Вычислить промежуток времени между двумя нажатиями клавиши. Нажатие функциональных клавиш игнорировать. Если нажата функциональная клавиша, предложить нажать другую клавишу. Результат вывести на экран
48	Ввести три последовательности символов. Ввести символ и найти его в каждой последовательности. На экран вывести сообщение, каким по счету является введенный символ в каждой последовательности
49	Ввести строку и символ с клавиатуры. Посчитать сколько раз введенный символ встречается в исходной последовательности
50	Сложить два числа, введенных с клавиатуры. Результат умножить на разность этих чисел

51	Ввести строку и символ с клавиатуры и вставить символ в строку после каждого символа
52	Ввести строку и символ с клавиатуры. Проверить, есть ли такой символ в строке. Если такой символ встречается не один раз, поместить их все в начало строки
53	Ввести символ с клавиатуры. Найти символ в исходной последовательности и сосчитать его порядковый номер
54	Ввести три последовательности символов. Сосчитать количество символов в каждой из последовательностей
55	Ввести символ с клавиатуры. Найти, в какой из трех введенных последовательностей есть введенный символ
56	Ввести символ и строку с клавиатуры. Проверить, есть ли такой символ в строке. Если такого символа нет, заменить им три последних элемента введенной строки
57	Ввести строку и символ с клавиатуры. Проверить есть ли такой символ в строке. Если нет, заменить все символы исходной последовательности введенным символом
58	Вести строку символов с клавиатуры, используя прерывание 21h с различными функциями: – функция 01h с эхо; – функция 07h без эхо; – функция 0Ah буферизованный ввод

3. ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ФАЙЛОВОЙ СИСТЕМЫ И СИСТЕМНЫЕ СРЕДСТВА ОБСЛУЖИВАНИЯ ДИСКОВ И ФАЙЛОВ

3.1. Распределение дискового пространства

Самый первый сектор жесткого диска (сектор 1 стороны 0 цилиндра 0) содержит главную загрузочную запись (Master boot), занимающую один сектор и включающую в себя часть программы начальной загрузки и таблицу разделов диска. В таблице разделов указываются адреса и размеры разделов, на которые разбит диск. Всего в таблице разделов зарезервировано место для четырех записей о разделах. DOS предоставляет возможность создания не более двух разделов. Один из них, называемый первичным, служит для размещения системных файлов и является загрузочным диском C:. Второй раздел называется расширенным, в нем можно создать один или несколько логических дисков (рис. 3.1).

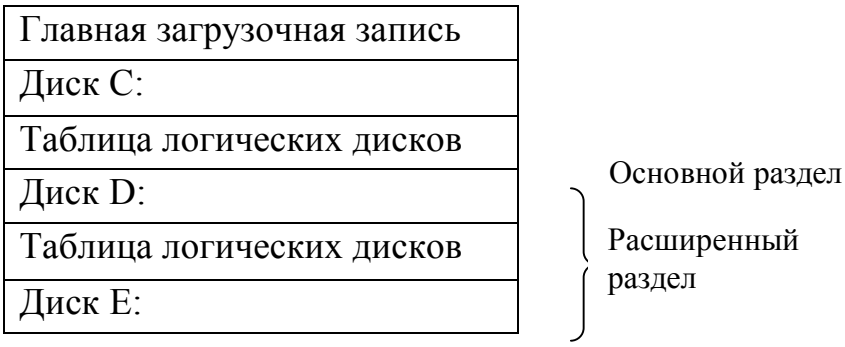

Рис. 3.1. Структура жесткого диска

Таким образом, жесткий диск разбивается на логические диски, каждый из которых занимает целое число цилиндров. Каждому логическому диску, входящему в расширенный раздел, предшествует сектор, содержащий таблицу логических дисков. В этой таблице указываются адреса и размеры данного и следующего логических дисков. Таблица логических дисков располагается в самом первом секторе области, выделенной под логический диск.

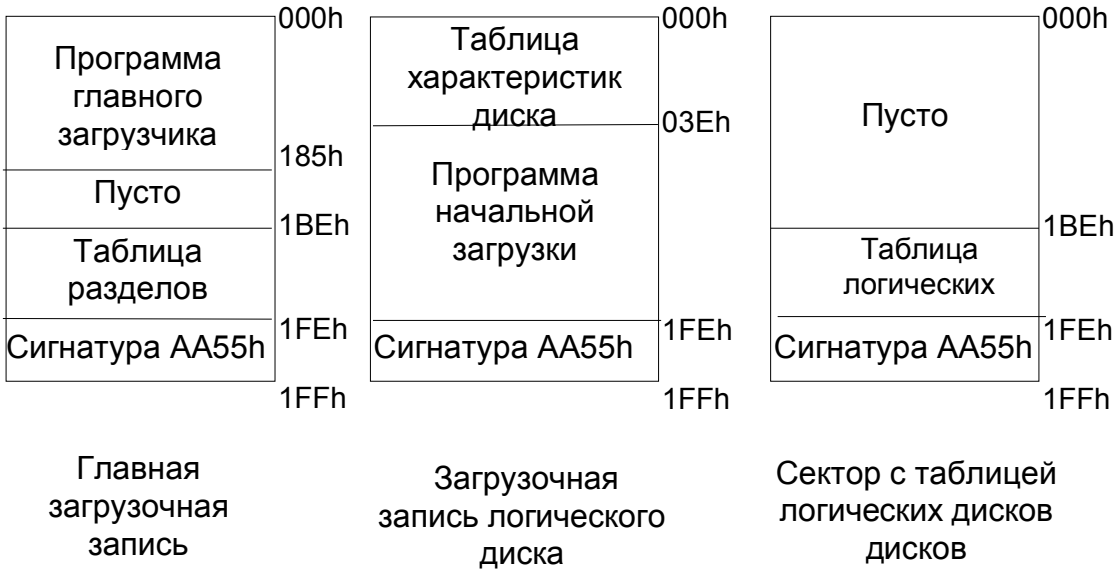

Рис. 3.2. Системные секторы

Каждый логический диск имеет свою относительную нумерацию секторов, начинающуюся от 0. Сектор с таблицей логических дисков, как и сектор главного загрузчика, не входит в систему относительной нумерации секторов и в этом смысле не принадлежит логическому сектору.

Четыре области логического диска – запись начальной загрузки, таблица распределения дискового пространства, основной каталог и область для данных – размещаются последовательно, начиная с сектора 0 (дорожка 0, сектор 1). Для различных видов дисков размер областей меняется, но их расположение и структура сохраняются.

Таким образом, жесткий диск разбивается на логические диски, каждый из которых занимает целое число цилиндров. Каждому логическому диску, входящему в расширенный раздел, предшествует сектор, содержащий таблицу логических дисков. В этой таблице указываются адреса и размеры данного и следующего логических дисков. Таблица логических дисков располагается в самом первом секторе области, выделенной под логический диск.

Загрузочный сектор и зарезервированные сектора
Первая копия FAT
Вторая копия FAT
Корневой каталог
Область данных

Рис. 3.3. Области логического диска

3.1.1 Запись начальной загрузки

Запись начальной загрузки имеют все диски, даже если на них нет операционной системы. Запись начальной загрузки всегда имеет длину один

сектор и занимает сектор 0. Основное, что в нем содержится, – это короткая программа, которая запускает процесс загрузки операционной системы с диска в память. Программа начинается с первых трех байтов сектора. Байты 1 и 0 содержат инструкцию для перехода к исполнимой части программы (обычно jmp 2b), а байт 2 содержит 90h (NOP). Пространство между двумя частями программы содержит параметрическую информацию о диске (табл. 3.1, в скобках указана информация для формата 1.44M). Последние два байта записи содержат индикацию 55 AA.

Таблица 3.1

Параметрическая информация о диске

Смещение	Длина	Содержание
00h	3 байта	Команда перехода на программу начальной загрузки
03h	8 байт	Идентификатор изготовителя (MSDOS 5.0)
0Bh	1 слово	Размер сектора в байтах (512)

Окончание табл. 3.1

Смещение	Длина	Содержание
0Dh	1 байт	Размер кластера в секторах (1)
0Eh	1 слово	Кол-во зарезервированных секторов в начале (1)
10h	1 байт	Количество копий FAT (2)
11h	1 слово	Количество элементов основного каталога (E0h)
13h	1 слово	Общее количество секторов диска (2880)
15h	1 байт	Идентификатор формата (F0h)
16h	1 слово	Размер FAT в секторах (9)
18h	1 слово	Количество секторов на дорожке (18)
1Ah	1 слово	Количество поверхностей (головок) (2)
1Ch	4 байта	Количество скрытых секторов
27h	4 байта	Серийный номер тома
2Bh	11 байт	Метка тома

FAT32 дисководы содержат более зарезервированных секторов, чем дисководы с FAT12 или с FAT16. Число зарезервированных секторов – обычно 32, но может измениться.

Так как в FAT32 Блок Параметров BIOS (BPB) имеет больший размер, чем стандартный BPB, то блок начальной загрузки на дисководах FAT32 занимает больше, чем 1 сектор. Кроме того, имеется сектор в зарезервированной области на дисководах FAT32, который содержит значения для количества свободных кластеров и номера кластера, распределенного последним. Эти дополнительные поля позволяют системе инициализировать значения без необходимости чтения всей таблицы размещения файлов.

3.1.2. Таблица распределения дискового пространства

Таблица распределения дискового пространства (File Allocation table – FAT) содержит информацию о расположении файлов, свободном пространстве на дисках и неисправных блоках, а также идентификатор формата диска. Для каждого файла поддерживается цепочка элементов FAT, каждый из которых указывает область фиксированной длины, занимаемой файлом на диске. В каталоге, содержащем файл, есть указатель к началу цепочки. При удалении файла элементы и соответствующие области диска освобождаются и могут быть использованы для другого файла. Основное преимущество этой организации заключается в возможности не только последовательного, но и прямого доступа к данным файла. Недостатком является постепенное фрагментирование диска при активной работе с файловой системой (создание и удаление файлов) – файлы не занимают непрерывные области, фрагментирование значительно снижает быстродействие. Поскольку FAT содержит очень важную для надежности данных информацию, то на диске хранятся две копии FAT.

Каждый элемент FAT (за исключением первых двух) соответствует определенному кластеру диска. Кластер – это группа стандартных секторов размером 512 байт. Группировка секторов необходима, чтобы уменьшить размер FAT. Однако большие кластеры, используемые на фиксированном диске, напрасно расходуют дисковое пространство при записи маленьких файлов. Первые два элемента FAT (3 или 4 байта) используются в качестве идентификатора формата диска. Значащим является только первый байт, остальные всегда содержат FFh. Для дискеты 1.44M идентификатор F0, для жесткого диска – F8.

Поскольку первые два элемента FAT (0 и 1) используются идентификатором формата диска, то нумерация фактических элементов начинается с 2. Нумерацию кластеров также принято начинать с 2. Каждый элемент FAT содержит число, которое идентифицирует соответствующий кластер или как свободный, или включенный в файл, или неиспользуемый, или зарезервированный для специальных целей.

Длина элементов FAT 12, 16 или 32 бита. Использование 12-битовых элементов FAT преследует цель максимально уменьшить размер FAT и используется на дискетах. В данном случае значения двух соседних элементов сохраняются в трех последовательных байтах. Это усложняет прослеживание цепочки элементов, но работу с FAT обычно берет на себя операционная система, которая осуществляет необходимые вычисления. Существуют, однако, и специальные случаи, когда необходим прямой доступ к FAT. В этих случаях для нахождения следующего элемента FAT выполняют следующие вычисления:

1) умножается номер кластера из текущего элемента FAT на 1,5;
2) результат (округленный к меньшему целому числу) используется как смещение в FAT, откуда считываются два байта;
3) следующий элемент содержится в первых 12 битах при четном номере кластера или в последних битах при нечетном номере.

Для преобразования номера кластера в номер сектора нужно выполнить следующее:

1) из номера кластера вычитается 2;

2) результат умножается на количество секторов в одном кластере;
3) к полученному результату добавляется количество секторов, занимаемых системными областями.

В том случае, когда количество секторов больше 4080, используется FAT с 16-битовыми элементами. Специальные значения элементов FAT, такие как дефектный блок, последний элемент цепочки FAT и т.д. являются логическим расширением значений 12-битового варианта – просто добавляется шестнадцатеричное F в начале. При поиске следующего элемента в цепочке FAT сложные вычисления уже не нужны, так как элементы имеют длину 2 байта.

Следует обратить внимание, что старшие 4 бита 32-разрядных значений в FAT32 таблице размещения файлов зарезервированы и не являются частью номера кластера. Приложения, которые непосредственно читают FAT32 таблицу размещения файлов, должны маскировать эти биты и сохранять их при записи новых значений.

В элементе каталога для файла FILEN.TXT указан номер первого кластера, распределенного файлу 17h. В этом случае цепочка кластеров для этого файла будет выглядеть так, как показано на рис. 3.4

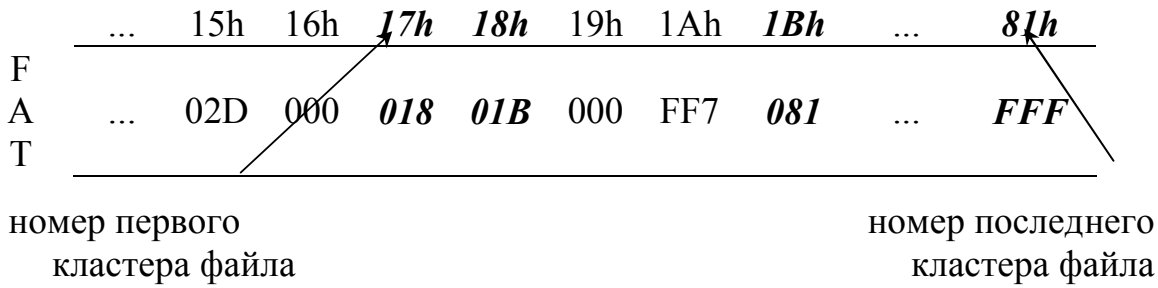

Рис. 3.4. Цепочка кластеров, распределенных файлу

3.1.3. Каталоги

Основной (корневой) каталог – это начало древовидной файловой структуры диска. Как и все остальные каталоги, создаваемые на диске, он содержит информацию о файлах, которые ему принадлежат. Наиболее существенная часть этой информации – имя файла и указатель к началу соответствующей цепочки элементов FAT. После форматирования диска корневой каталог либо пуст, либо содержит системные файлы DOS. Пользователь строит файловую структуру диска, добавляя к основному каталогу файлы или новые каталоги. С точки зрения файловой организации каталоги тоже являются файлами. Исключением является корневой каталог, который находится в определенном месте на диске, имеет фиксированную длину и который нельзя удалить.

Корневой каталог на дисководе FAT32 не имеет фиксированного расположения, как на дисководах с FAT12 и с FAT16. На дисководах с FAT32, корневой каталог – обычная цепочка кластеров. Элемент в структуре BPB FAT32 содержит номер первого кластера в корневом каталоге. Это позволяет корневому каталогу расти так, как необходимо.

Для каждого файла на диске имеется один элемент в определенном каталоге. Один элемент корневого каталога выделяется для метки диска. Для каждого каталога имеется элемент в его родительском каталоге. Кроме того, каждый каталог, за исключением корневого, содержит по одному элементу для специальных имен "." и "..". Эти элементы указывают начало цепочки в FAT соответственно для каталога и для его родительского каталога.

Каждый элемент каталога имеет длину 32 байта и состоит из восьми полей (табл. 3.2). Для FAT32 элемент каталога отличается длиной номера начального кластера, поэтому структура его элементов имеет несколько иной вид (табл. 3.3).

Таблица 3.2

Элемент каталога FAT12 и FAT16

Смещение	Число байтов	Содержание
00h	8	Имя файла в кодах ASCII
08h	3	Расширение имени файла в кодах ASCII
0Bh	1	Байт атрибутов файла
0Ch	10	Зарезервировано
16h	2	Время создания или последней модификации
18h	2	Дата создания или последней модификации
1Ah	2	Номер начального кластера файла
1Ch	4	Фактическая длина файла в байтах

Первый байт поля имени используется для обозначения трех специальных случаев:

- значение 00H в первом байте показывает, что этот элемент каталога никогда не был использован. Так как каталог заполняется последовательно, это означает, что и следующие за ним элементы не были использованы;
- при удалении файла DOS записывает E5H в первом байте соответствующего элемента каталога. Все остальные байты элемента остаются без изменения. Элемент становится свободным, но после него могут быть другие активные элементы. При удалении файла DOS освобождает и все элементы соответствующей цепочки в FAT. Но сам файл продолжает существовать на диске, что позволяет соответствующим программам восстановить удаленный файл, если элемент каталога или кластеры файла в области данных не используются уже другим файлом;
- значение 2EH (символ ".") в первом байте показывает, что этот элемент служит для описания самого каталога. Если и второй байт содержит 2EH, элемент описывает родительский каталог ("..").

Таблица 3.3

Элемент каталога FAT32

Смещение	Размер	Значение
0	8	Имя файла в кодах ASCII
8	3	Расширение имени файла в кодах ASCII

11	1	Атрибут
12	8	Зарезервировано
20	2	Номер начального кластера (старшие разряды)
22	2	Время создания или последней модификации
24	2	Дата создания или последней модификации
26	2	Номер начального кластера (младшие разряды)
28	4	Фактическая длина файла в байтах

Для файлов, которым не выделено места, и для метки тома поле номера первого кластера содержит 0000H.

Байт атрибута определяет каждый файл, например, как системный или скрытый и т.д. Значение битов байта атрибутов приведены в табл. 3.4.

Таблица 3.4

Назначение битов байта атрибутов

7 6 5 4 3 2 1 0	ЗНАЧЕНИЕ	НАЗНАЧЕНИЕ
. 1	1h	защищенный
. . . . . 1 .	2h	скрытый
. . . . . 1 . .	4h	системный
. . . . 1 . . .	8h	метка тома
. . . 1	10h	каталог
. . 1	20h	архивный
. 0	40h	не используется
0	80h	не используется

Когда программа отправляет запрос к операционной системе с требованием предоставить ей содержимое какого-либо файла, ОС просматривает запись каталога для него, чтобы найти первый кластер этого файла. Затем она обращается к элементу FAT для данного кластера, чтобы найти следующий кластер в цепочке. Повторяя этот процесс, пока не обнаружит последний кластер файла, ОС определяет, какие кластеры принадлежат данному файлу. Таким образом, система может предоставить программе любую часть запрашиваемого ею файла.

3.1.4. Хранение длинных имен

Требования совместимости, которым должна удовлетворять система Windows, означают, что невозможно просто изменить существующий формат хранения данных на диске, который применяется в FAT файловой системе.

VFAT файловая система поддерживает как длинные, так и короткие имена файла. 32-байтный элемент каталога идентичен тому формату, который поддерживают предыдущие версии MS-DOS.

Метод работы с длинными именами файлов строится на использовании байта атрибута элемента каталога для кроткого имени файла. Установка младших четырех битов этого байта (значение 0Fh) задает элементу каталога атрибуты “только для чтения”, “скрытый”, “системный” и “метка тома”. Добавление

атрибута “метка тома” дает не имеющее смысл сочетание и защищает элемент каталога от изменения.

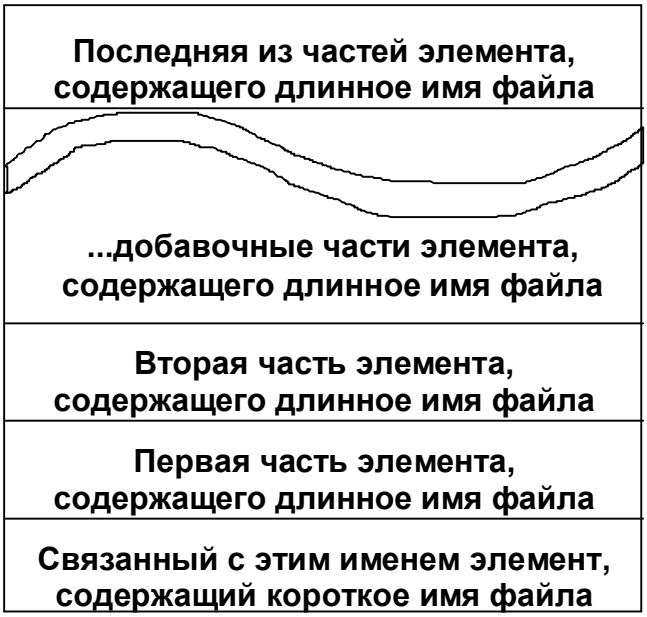

Рис. 3.5. Формирование длинного имени в каталоге из элементов коротких имен

Для того чтобы исключить возможность, что какая-нибудь утилита обслуживания диска уничтожит данные, разработчики ввели в API функцию “исключительного закрепления метки тома” (exclusive volume lock), которую приложение должно вызвать прежде, чем Windows позволит ему осуществить прямую запись на диск (при помощи MS-DOS прерываний Int 13h и Int 26h).

Windows использует для формирования длинного имени несколько последовательных коротких имен – элементов каталога (рис. 3.5), защищая каждое из 32-байтных имен при помощи атрибута 0Fh. В пределах одного кластера, в котором хранится информация о содержимом каталога, элемент с длинным именем файла располагается в соответствии с форматом, который показан на рисунке 3.5. Длинное имя файла не может существовать без связанного с ним элемента с коротким именем. Если имеет место такая ситуация, значит, нарушена целостность данных на диске.

Каждый 32-байтный элемент, описывающий длинное имя файла, содержит *порядковый номер, защитный байт атрибута, значение типа и контрольную сумму* (рис. 3.6). *Порядковый номер* помогает Windows узнать о непоследовательном или некорректном изменении структуры каталога. Поле *типа* идентифицирует элемент как LONG_NAME_COMP (элемент длинного имени) либо как LONG_CLASS (32-байтный элемент, который содержит информацию о классе для данного файла). Если элемент представляет собой часть имени, то большая часть 32 байт используется для хранения символов имени. Если это элемент, описывающий класс, то в нем хранится информация о классе. Система хранит длинные имена файлов в виде символов таблицы Unicode, т.е. каждый символ имени файла требует 16 бит. *Контрольная сумма* вычисляется по связанному с данным файлом короткому имени. Если короткое имя изменится вне

среды Windows, то Windows поймет, что элементы длинного имени больше не имеют смысла.

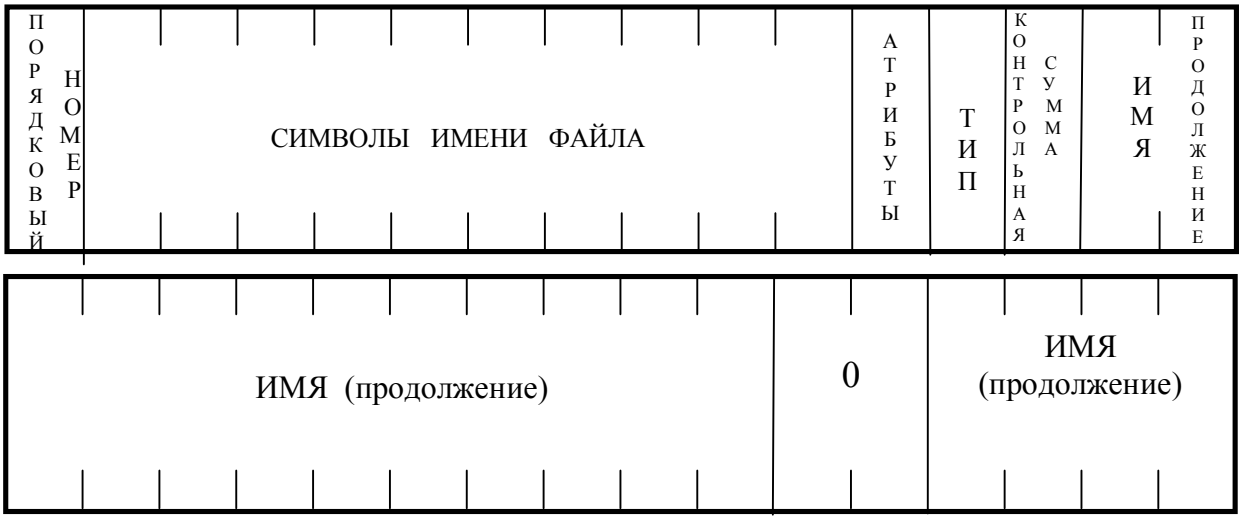

Рис.3.6. Формат элемента каталога, в котором хранится длинное имя файла

Основная проблема при формировании короткого имени, связанного с длинным, состоит в создании уникального короткого имени, которое не совпадает с каким-нибудь уже существующим коротким именем.

3.2. Примеры программ

В данном разделе не будут приводиться детальные описания используемых функций и сервисов операционных систем. Здесь будут показаны лишь отличия между некоторыми интерфейсами доступа к дисковым устройствам. В качестве примера, на котором будут показаны отличия, взята задача чтения первого сектора диска.

3.2.1. Работа в DOS

В данном разделе будет рассмотрена работа с дисками в реальном режиме как с использование прерываний (сервисов) BIOS, так и прерываний (сервисов) DOS.

Прерывание BIOS int 13h

Это один из самых старых и, следовательно, самых низкоуровневых интерфейсов. В классическом варианте использует систему адресации цилиндр-головка-сектор, или C-H-S (Cylinder-Head-Sector). Данная трехмерная система координат сектора на диске отражает физическую структуру диска. Дисковый накопитель состоит из нескольких, насаженных на общий стержень, магнитных поверхностей – «блинов». Таким образом, существует несколько поверхностей, на которые могут быть записаны данные. Запись на каждую поверхность производит магнитная головка (координата H). Данные на поверхность при вращении всего комплекта «блинов» заносятся по замкнутой окружности – дорожке (или цилиндру – комплект дорожек), расположенных на разных поверхностях друг над другом (координата C). Количество дорожек на всех поверхностях одинаково. И

наконец, каждая дорожка разбита на некоторое количество секторов (S-координата). В каждом секторе сохраняется объем полезных данных в 512 байт. Сектора начинаются с единицы, а цилиндры и головки – с нуля.

Таким образом, каждый сектор можно задать трехразрядным числом: C-H-S. Сектор – самый младший «разряд» этого числа; перенос из данного разряда – это переход на следующую поверхность (головку) внутри цилиндра. Когда исчерпаны все головки – происходит «перенос разряда» в следующий цилиндр. Перевести число из такой системы счисления в любую другую достаточно просто:

$$N = C*NH*NS + H*NS + S,$$

где NH – число сторон (головок);
NS – количество секторов на дорожке.

Соответственно и обратное преобразование не составляет труда: имея N, делим его на NS нацело, и получаем в остатке S (номер сектора будет S+1), а в частном N2 = C*NH + H. Делим нацело N2 на NH, и в остатке получаем H, а в частном – C.

Гораздо проще рассматривать диск не как набор из цилиндров, головок и секторов, а как линейный одномерный массив секторов. В настоящее время именно так дисковый накопитель и рассматривают, и такой вид адресации, получивший название LBA, поддерживается на уровне контроллера диска.

В примере показано чтение первого сектора (0-0-1) средствами BIOS:

```
.model small
.stack 100h
.data

; опишем структуру для доступа к логическому диску
DRVPatam struc
        c  dw ?  ; максимальный номер циллиндра (Cylinder)
        h  dw ?  ; максимальный номер головки (Head)
        s  dw ?  ; максимальный номер сектора (Sector)
DRVPatam ends

        ; блок параметров и буфер данных
        block DRVPatam <?, ?, ?>
        buf  db 512 dup (?)

.code
.486
main:
        mov ax, @data
        mov ds, ax

        ; для начала следует определить параметры диска...
        ; для преобразований номера сектора в координаты C-H-S
```

```
        ; собственно этого-то мы делать и не будем...
        ; (просто считаем первый (0-0-1) сектор!)
        mov ah, 08h ; функция
        mov dl, 00h ; 00h-7Fh – гибкий диск, 80h-0FFh – жесткий диск
        int 13h

        mov al, ch ; выделим количество цилиндров (дорожек)
        mov ah, cl
        shr ah, 6
        inc ax
        mov block.c, ax

        and cx, 0111111b ; выделим количество секторов
        mov block.s, cx

        xor ah, ah ; количество головок (сторон)
        mov al, dh
        inc ax
        mov block.h, ax
        ; итак, читаем первый сектор N=1 CHS(N)=0-0-1
        mov ah, 2h  ; вторая функция биос "тринадцать аш" – чтение сектора
        mov al, 1   ; количество считываемых секторов
        mov ch, 0   ; номер циллиндра C
        mov dh, 0   ; номер головки   H
        mov cl, 1   ; номер сектора   S
        mov dl, 00h ; 00h-7Fh – гибкий диск, 80h-0FFh – жесткий диск
        push ds
        pop  es     ; es=ds ! буфер должен быть по адресу ES:BX
        lea bx, buf
        int 13h
        jc error

        ; успешно считан первый сектор дискеты!
error:
        mov ax, 4c00h
        int 21h
end main
```

Как уже отмечалось, проще рассматривать диск как линейный одномерный массив секторов. BIOS поддерживает и такой интерфейс. Он был предложен Phoenix Technologies и стал стандартом. Для того чтобы создать метод независимый от геометрии жесткого диска пришлось разработать унифицированный формат данных, который был назван пакетом дискового адреса. Теперь все адресные данные записываются в пакет – область памяти, а не

в регистры процессора, как было ранее. Теперь сервису передается только указатель на пакет.

Примечание: этот способ чтения неприменим для дискеты, так как воспринимаются только EDD/ATA устройства – жесткие диски и НГМД большой емкости.

В следующем примере показано чтение первого сектора диска расширенными средствами BIOS:

```
.model small
.stack 100h
.data

; опишем структуру для доступа к логическому диску
DRVPacket struc
        packetSize  db ? ; размер пакета в байтах
        reserved1   db 0 ; резерв №1
        nBlocks     db ? ; количество передаваемых блоков:
                                ; если 254 мало – пишем сюда
                    ; 255(0FFh) и сколько
                    ; надо заносим в nBlocksEx,
                    ; если и 2^32-2 мало – вы в
                    ; далеком будущем!
        reserved2   db 0 ; резерв №1
        offs        dw ? ; смещение – часть дальнего адреса
                                ; буфера
        segm        dw ? ; сегмент – часть дальнего адреса
                                ; буфера
        LBA         dq 0 ; 64 разрядный LBA 2^73 байт! или
                                ; 2^43 гигабайт
        addr64      dq ? ; 64 разрядный линейный адрес буфера
                                ; передачи!!! –
                                ; BIOS вечен – он никуда не денется
                                ; и в 64-х разрядных версиях
                                ; процессоров, но будет иметь 3D
                                ; графический интерфейс
                                ; и станет понимать речь...
        nBlocksEx   dd ? ; если в nBlocks количество блоков
                                ; для передачи не входит,
                    ; то... см. выше...
        reserved3   dw 0 ; резерв №3
DRVPacket ends

        ; блок параметров и буфер данных
        block DRVPacket <?>
        buf  db 512 dup (?)
```

```
.code
.486
main:
        mov ax, @data
        mov ds, ax

        ; определим параметры...
        mov block.packetSize, 32 ; размер структуры пакета
                                                ; дискового адреса
        mov block.offs, offset buf ; сегментная часть адреса
                                                        ;буфера
        mov block.segm, ds     ; смещение буфеера в block.segm
        mov block.nBlocks, 1         ; считаем один сектор
        mov dword ptr block.LBA[0],0 ; пока 32 разрядный LBA
                        ; первого (нулевого!) блока
                                                ;(сектора)

        mov ah, 42h                     ; 42h – функция!
        mov dl, 80h                 ; 80h-83h – жесткий диск
        lea si, block               ; DS:SI – адрес пакета
                                        ; дискового адреса
        int 13h                         ; запуск сервиса (надо бы
                                        ; проверить вначале ;поддерживаются ли
                                    ; дополнительные функции)
        test ah, ah             ; вернула не ноль – непорядок!
        jnz error

        ; успешно считан первый сектор с винчестера!
error:
        mov ax, 4c00h
        int 21h
end main
```

Считать/записать сектор можно и с помощью прерываний DOS 25h (прямое чтение) и 26h (прямая запись). Тонкость в использовании заключается в том, что сервис различает не физические устройства, а разделы на диске (логические диски), которые нумерует с нуля (0-A:, 1-B:, 2-C:) и, кроме того, в силу недостатков интерфейса (см. пример) не может адресовать больше 32 мегабайт в логическом диске:

Пример низкоуровневого обращения к диску средствами DOS для абсолютного чтения/записи

```
;читаем первый сектор посредством int 25h с дискеты
.model small
.stack 100h
.data
        buf  db 512 dup (?)

.code
.486
main:
        mov ax, @data
        mov ds, ax

        mov al, 00h         ; AL==00h логический диск 0-A:,
                                    ; 1-B:,
                                    ; 2-C:,...
        mov cx, 1           ; число считываемых секторов
        mov dx, 0           ; номер начального сектора
                                    ; 2^16*2^9=2^25=2^5
                    ; мегабайт (32 мегабайта максимум!)
        lea bx, buf   ; DS:BX – адрес буфера
        int 25h       ; вызываем сервис
        jc error

        ; первый сектор считан успешно!
error:
        mov ax, 4c00h

        int 21h
end main
```

Лишен недостатков сервис DOS для работы с разделами большого размера. Под большим размером понимается ограничение в 2048 гигабайт. Подход тот же, что и в случае с BIOS – создать блок параметров операции и передать сервису адрес:

Пример низкоуровневого обращения к диску средствами DOS для абсолютного чтения/записи – работа с разделами большого объема.

```
.model small
.stack 100h
.data

; опишем структуру для доступа к логическому диску
INT25Block struc
        sectorno  dd ?  ; номер сектора 2^32 секторов
```

```
                ; 2^32*2^9=2^41=2^11 гигабайт (2048 гигабайт...)
        nsectors  dw ?  ; количество считываемых секторов
        offs      dw ?  ; смещение – часть дальнего указателя на буфер
        segm      dw ?  ; сегмент – часть дальнего указателя на буфер
INT25Block ends

        ; блок параметров и буфер данных
        block INT25Block <?, ?, ?, ?>
        buf  db 512 dup (?)

.code
.486
main:
        mov ax, @data
        mov ds, ax

        mov block.sectorno, 0 ; номер считываемого сектора
        mov block.nsectors, 1 ; количество считываемых секторов
        mov block.offs, offset buf  ; смещение дальнего указателя на буфер
        mov block.segm, ds          ; сегмент дальнего указателя на буфер

        mov al, 02h        ; AL==00h логический диск 0-A:, 1-B:, 2-C:,...
        mov cx, 0FFFFh     ; признак расширенной функции
        lea bx, block      ; DS:BX – адрес блока описания операции чтения
        int 25h            ; вызываем сервис
        jc error

        ; успешно считан первый сектор диска C:!
error:
        mov ax, 4c00h
        int 21h
end main
```

3.2.2. Работа в Windows

Для работы с дисками на низком уровне следует учитывать версию OS Windows, так как в процессе развития этих операционных систем менялись подходы к реализации данных возможностей.

Windows 95/98/ME

Низкоуровневые операции работы с диском в этих операционных системах производятся через драйвер виртуального устройства “VWin32.VXD”. «*.VXD» – Virtual X Device (Virtual Device Driver). VxD-драйвер – это драйвер какого-либо

устройства, работающий в привилегированном режиме микропроцессора. Он может расширять возможности сервисов ядра Windows, контролировать доступ к аппаратуре или выполнять обе эти функции. Согласно документации MSDN, в этих ОС корректно работает только функция чтения сектора логического диска (не физического, как в случае с MS-DOS). В связи с этим в примере будет произведено считывание 0-го сектора диска C: (в программе предполагается, что диск C: имеет файловую систему FAT32; на экран будет выведена информация из этого сектора).

Выполнение обращения к дисковому устройству можно разделить на три этапа:

1) открытие виртуального устройства “VWin32” с помощью функции “CreateFile”;
2) выполнение операции чтения сектора с помощью функции “DeviceIOControl”;
3) вывод содержимого сектора на экран.

Приведенный пример будет работать только под управлением ОС Windows 95/98/ME.

Функция CreateFile позволяет получить дескриптор (hDevice) VxD-драйвера, который будет использоваться функцией DeviceIoControl.

Описание аргументов данной функции уже давалось выше, поэтому необходимо рассмотреть лишь то, как следует указать имя VxD устройства ‘\\.\VxDName’ (в данном случае ‘\\.\VWin32’).

Обратиться к драйверу устройства позволяет функция

```
BOOL DeviceIoControl(
  HANDLE hDevice, ;дескриптор VxD – возвращенный
                          ;CreateFile
  DWORD dwIoControlCode,//код выполняемой драйвером
                // операции. Специфичен
                // для каждого конкретного VxD
  LPVOID lpInBuffer,  //указатель на входные данные
          //(формат и размер специфичны для VxD!)
  DWORD nInBufferSize,// размер буфера входных
                 // данных lpInBuffer в байтах
  LPVOID lpOutBuffer,                  //указатель на выходные //данные
  DWORD nOutBufferSize, // размер буфера выходных
                 // данных lpInBuffer в байтах
  LPDWORD lpBytesReturned,        //указатель на двойное
```

```
                    // слово (после вызова
                                //там количество полученных байт)
  LPOVERLAPPED lpOverlapped // указатель на
          // структуру OVERLAPPED,необходимую
                                // для асинхронного ввода вывода
          //то есть вызвавший операцию процесс
          // не дожидается окончания ввода-
          // вывода, а продолжает выполнение
          // (система уведомит процесс об
          // окончании операции послав
                                //определенное пользователем событие)
  );
        push  0
        push  offset countb
        push  DEVIOCTL_REGISTERS_
        push  offset reg_EBX
        push  DEVIOCTL_REGISTERS_
        push  offset reg_EBX
        push  VWIN32_DIOC_DOS_DRIVEINFO
        push  eax
        call  DeviceIoControl                    ; stdcall
```

Кратко опишем возможности драйвера VWIN32.VXD.

Задавая параметр dwIoControlCode в функции, можно определить, к какому сервису BIOS или DOS идет обращение.

dwIoControlCode	Выполняемое VWin32 действие
VWIN32_DIOC_DOS_DRIVEINFO	Выполнить эквивалент int 21h подмножество функций 730Xh DOS
VWIN32_DIOC_DOS_INT13	Выполнить эквивалент int 13h – доступ к диску BIOS
VWIN32_DIOC_DOS_INT25	Выполнить эквивалент int 25h – абсолютное чтение с диска DOS
VWIN32_DIOC_DOS_INT26	Выполнить эквивалент int 25h – абсолютная запись на диск DOS
VWIN32_DIOC_DOS_IOCTL	Выполнить эквивалент определенной DOS функции ввода вывода (int 21h функции 4400h-4411h)

Параметры lpInBuffer и должны lpOutBuffer ссылаться на структуру

```
typedef struct _DIOC_REGISTERS {
   DWORD reg_EBX; //регистр процессора EBX и т.д.
   DWORD reg_EDX;
   DWORD reg_ECX;
   DWORD reg_EAX;
   DWORD reg_EDI;
   DWORD reg_ESI;
   DWORD reg_Flags; //регистр флагов
};
```

Для вызова соответствующих сервисов надо ознакомится с их описанием и сформировать в структуре _DIOC_REGISTERS состояние регистров, которое требовалась по спецификации для вызова сервиса.

Следует отметить некоторые особенности, связанные с отличиями в адресации в реальном режиме, используемом DOS, и защищенном режиме со страничной адресацией, используемом Windows. В некоторых случаях в сервисах DOS (BIOS) требуется задавать дальние адреса структур в регистрах, например ES:BX, а в структуре _DIOC_REGISTERS сегментные регистры не присутствуют вообще. Дело в том, что Windows, используя страничную адресацию, получает «плоскую» четырех-гигабайтную память и базы всех дескрипторов в страничном режиме настроены на начало памяти (на 0). В результате, по какому бы сегментному регистру ни приходилось обращаться к памяти, результат будет один, потому что адрес определяется только смещением. Поэтому, когда в описании сервиса реального режима требуется, например, указать адрес буфера в паре ES:BX, нужно записать в поле reg_EBX смещение буфера в защищенном режиме.

Не следует забывать выгружать драйвер после использования функцией CloseHandle(hDevice).

Пример низкоуровневого обращения к диску в Windows 95/98/ME

```
.386
.model flat, stdcall
locals
includelib ..\..\LIB\imp32i.lib
include ..\..\INCLUDE\w32.inc
```

```
VWIN32_DIOC_DOS_DRIVEINFO equ 6
CARRY_FLAG                equ 1

.DATA
sVWin32     db "\\.\VWin32",0
sMsg1       db "Could not open VWin32!!!",13,10,0
sMsg2       db "Could not read from disk!!!",13,10,0

countb      dd ?   ; сюда DeviceIOControl вернет количество
                   ; байт, записанных в структуру DEVIOCTL_REGISTERS

; опишем структуру, отражающую состояния регистров при обращении к IOCTL
;       DEVIOCTL_REGISTERS STRUC
reg_EBX     dd ?
reg_EDX     dd ?
reg_ECX     dd ?
reg_EAX     dd ?
reg_EDI     dd ?
reg_ESI     dd ?
reg_Flags   dd ?
;       DEVIOCTL_REGISTERS ENDS
DEVIOCTL_REGISTERS_     equ 4*7

; структура ввода вывода при работе с диском…
;       DISK_IO STRUC
StartSector  dd ?
Sectors      dw ?
Buffer       dd ?
;       DISK_IO ENDS

Boot        db 512 dup(?)

.CODE
Start:
        pushad
```

; **первый этап – открываем виртуальное устройство “VWin32”**

```
        push  0
        push  0
        push  OPEN_EXISTING
        push  0
        push  FILE_SHARE_READ or FILE_SHARE_WRITE
        push  GENERIC_READ
        push  offset sVWin32
```

```
        call    CreateFile                      ; stdcall
        cmp     eax, INVALID_HANDLE_VALUE  ; в случае ошибки
                                                ; CreateFile вернет
        jz      quit                            ; INVALID_HANDLE_VALUE

        mov     ebx, eax                ; сохраняем в ebx дескриптор
                                        ; открытого устройства

; второй этап – чтение сектора
        ; заполняем необходимые поля структур
        ; см. документацию на int 21h подмножество функций 730Xh DOS
        ;структура для чтения диска
        mov     StartSector, 0      ; номер сектора задаётся в формате LBA
        mov     Sectors, 1
        mov     Buffer, offset Boot

        ;заполняем регистры в соответствии с изложенными выше правилами
        ;и документацией на int 21h ax=730Xh
        mov     reg_EAX, 7305H
        mov     reg_EBX, offset StartSector ; на начало структуры DISK_IO
        mov     reg_ECX, -1
        mov     reg_EDX, 3                  ; 1=A:, 2=B:, 3=C:, ...

        ; формирование стека для вызова API функции DeviceIoControl
        push    0 ;нет асинхронного ввода/вывода–пустой указатель
        push    offset countb ;указатель на ячейку для количества считанных байт
        push    DEVIOCTL_REGISTERS_     ;размер входных данных
        push    offset reg_EBX          ;на начало структуры DEVIOCTL_REGISTERS
        push    DEVIOCTL_REGISTERS_     ;размер входных данных
        push    offset reg_EBX          ;на начало структуры DEVIOCTL_REGISTERS
        push    VWIN32_DIOC_DOS_DRIVEINFO;сервис IOCTL
        push    eax             ;дескриптор, полученный CreateFile
        call    DeviceIoControl         ; stdcall

        test    eax, eax                ; в случае ошибки функция
                                        ; DeviceIoControl
        jz      quit                    ; вернет false
        test    reg_Flags, CARRY_FLAG   ;в случае ошибки в reg_Fags
        jz      quit                    ; будет установлен флаг переноса

        ;мы успешно считали первый сектор – можно работать дальше!
quit:
        push    ebx
        call    CloseHandle ;закрываем устройство
```

```
        popad
        push  0
        call    ExitProcess ;выходим из программы
END Start
```

Windows NT/2000/XP/2003/Vista

В операционной системе Windows NT, а также в ОС построенных на основе Windows NT работа с диском производится через функции работы с файлами “CreateFile” и “ReadFile”, при этом пользователь, от лица которого выполняется обращение к диску, должен обладать правами администратора.

Выполнение программы можно разделить на три этапа:

1) открытие диска с помощью функции “CreateFile”;
2) выполнение операции чтения сектора с помощью функции “ReadFile”;
3) вывод содержимого таблицы разделов на экран.

Перечислим несколько особенностей при использовании функции CreateFile. В качестве имени создаваемого файла следует указать:

1) для того чтобы открыть жесткий диск:

«\\.\PHYSICALDRIVEx» – где x – номер жесткого диска. Нумерация дисков начинается с нуля. «\\.\PHYSICALDRIVE0» – нулевой (физически) жесткий диск.

2) для того чтобы открыть раздел диска или НГМД:

«\\.\A:» – вернет дескриптор гибкого диска

«\\.\C:» – вернет дескриптор раздела «C:»

Полученный дескриптор можно использовать в функциях ReadFile, WriteFile, интерпретируя диск как файл большого объема. Или использовать функцию DeviceIoControl, так как работа по-прежнему идет через драйвер устройства. Специфичные операции для данного драйвера рассматриваться не будут, в случае необходимости обращайтесь к MSDN или разделам справочников Windows SDK инструментальных сред Borland.

При использовании функций ReadFile, и WriteFile нет никаких отличий в работе с диском и файлом.

Полученный дескриптор по завершении работы следует закрыть функцией CloseHandle(hObject).

Дадим необходимую информацию по использованию функций SetFilePointer и ReadFile (WriteFile):

```
DWORD SetFilePointer(
  HANDLE hFile,        // дескриптор файла (в нашем случае диска) –
                       //результат работы CreateFile
```

```
  LONG lDistanceToMove,     // количество байт на которое следует
                            //передвинуть файловый указатель чт/зп
  PLONG lpDistanceToMoveHigh,  // если количество байт, на которое
                            // следует сдвинуть указатель больше 2^32,
                            //определить указатель на 64-битное смещение
                            //и передать в данном параметре указатель на него
  DWORD dwMoveMethod    // определяет относительно чего следует
                                  // сдвигать файловый указатель
      // FILE_BEGIN – смещение от начала (DistanceToMove воспринимается
      //как беззнаковое); FILE_CURRENT – относительно текущего
      // файлового указателя (DistanceToMove воспринимается как число со
знаком и знак
      //определяет куда (вперед + / назад -) будет происходить сдвиг); FILE_END-
      // смещение от конца (DistanceToMove воспринимается как беззнаковое)
);
```

Для чтения необходимо использовать функцию ReadFile. Она вернет ненулевое значение в случае успешного чтения. Прототип функции записи в целом аналогичен и не приводится

```
BOOL ReadFile(
  HANDLE hFile,         // дескриптор файла (в нашем случае диска) –
                  //результат работы CreateFile
  LPVOID lpBuffer,      // адрес буфера в памяти для чтения данных
  DWORD nNumberOfBytesToRead,       // количество считываемых байт
  LPDWORD lpNumberOfBytesRead,      // адрес ячейки для записи
                            // количества считанныхбайт
  LPOVERLAPPED lpOverlapped   // указатель на структуру
                  //  OVERLAPPED, необходимую для асинхронного ввода-//
                  вывода, то есть вызвавший операцию процесс не
                // дожидается окончания ввода-вывода, а
                // продолжает выполнение (система уведомит
                // процесс об окончании операции, послав
                // определенное пользователем событие)
);
```

Пример низкоуровневого обращения к диску

в Windows NT/2000/XP/2003/Vista

Приведенный пример будет работать только под управлением ОС Windows NT/2000/XP/2003/ Vista.

```
.386
.model flat, stdcall
locals
includelib ..\..\LIB\imp32i.lib
include ..\..\INCLUDE\w32.inc

PARTENTRY STRUC
  flag          db ?
  beg_head          db ?
  beg_sec_cyl       dw ?
  sys               db ?
  end_head          db ?
  end_sec_cyl       dw ?
  rel_sec           dd ?
  part_size         dd ?
PARTENTRY ENDS
PARTENTRY_    equ 1+1+2+1+1+2+4+4

.DATA
sDrive        db "\\.\PHYSICALDRIVE0",0 ; будем производить
                                        ; чтение с
                                        ; физического диска

.DATA?
BytesRead   dd ?                ; сюда функция “ReadFile” вернет
                                        ; количество прочитанных байт,
Boot          db 512 dup(?)     ; а сюда прочитанные данные

.CODE
Start:
        pushad

        ; первый этап – открытие диска
        push  0
        push  0
        push  OPEN_EXISTING
        push  0
        push  FILE_SHARE_READ or FILE_SHARE_WRITE ;
        ;обязательно разрешить остальным писать на диск, а то
```

```
        ;как же ОС будет работать?
        push  GENERIC_READ; открываем диск на чтение
        push  offset sDrive
        call    CreateFile    ; stdcall
        cmp   eax, INVALID_HANDLE_VALUE ; в случае ошибки
                                              ;“CreateFile” вернет ; ;
                                              ;INVALID_HANDLE_VALUE
        jz      quit

        mov   ebx, eax      ; сохраняем в ebx дескриптор второй этап –
                            ; выполняем чтение с диска
        ; позиционируемся в начало диска
        push FILE_BEGIN ;от начала файла ДИСКА!;)
        push 0 ;смещаемся на 0 от начала
        push 0 ;указателя нет – нам хватает и 2^32
        push eax ;дескриптор файла ДИСКА!;)
        call    SetFilePointer

        ; выполняем чтение с диска
        push 0 ;нет необходимости в асинхронном в/выв
        push offset BytesRead ;ссылка на количество считанных байт
        push 512 ;считаем один сектор
        push offset Boot ;буфер – считаем сюда!
        push ebx ;дескриптор
        call    ReadFile
        test    eax, eax      ; в случае ошибки функция вернет false
        jz      quit
```

;успешно считан первый сектор – можно работать дальше!

```
quit:
        push  ebx
        call    CloseHandle ;закрыть указатель
        popad
        push 0
        call    ExitProcess
END Start
```

Вопросы для самопроверки

1. Главная загрузочная запись (MBR).
2. Таблица разделов.
3. Сколько разделов можно создать на жестком диске?
4. Первичный раздел жесткого диска.
5. Расширенный раздел жесткого диска.
6. Параметры логического диска.
7. Таблица логических дисков.

8. Области логического диска с файловой системой FAT.
9. Основные отличия файловых систем FAT12 и FAT16.
10. Особенности FAT32.
11. Фрагментация диска и причины ее возникновения.
12. Корневой каталог и его отличия от других каталогов.
13. Отличия корневого каталога в FAT32 и FAT12(16).
14. Определение кластера и сектора
15. В какой области логического диска сектора объединяются в кластеры?
16. Переход от номера кластера к номеру сектора.
17. Какая информация о файле находится в элементе каталога?
18. Алгоритм выделения места под файл с точки зрения файловой системы FAT.
19. Алгоритм обновления файла с точки зрения файловой системы FAT.
20. Алгоритм удаления файла с точки зрения файловой системы FAT.
21. Восстановление удаленного файла. Всегда возможно?
22. Перечислите особенности низкоуровневого обращения к диску в Windows 95/98/ME.
23. Перечислите особенности низкоуровневого обращения к диску в Windows NT/2000/XP/2003/Vista

Примерные задания для выполнения лабораторной работы

1	Найти на дискете в каталоге MYDIR первый файл типа .com. Вывести на экран его имя или сообщение, что такого файла нет
2	Открыть 2 входных файла и переписать четные записи из первого файла и нечетные записи из второго файла в выходной файл. О результате выполнения операции вывести сообщение на экран. Вывести размер выходного файла, используя запись о файле в каталоге
3	Получить атрибуты файла и установить атрибут «только для чтения», если он не установлен. Атрибут файла получить из записи в каталоге
4	Найти в каталоге все файлы с расширением .com. Вывести на экран список этих файлов, используя две цветовые палитры. Файлы искать с помощью чтения записей корневого каталога
5	Прочитать содержимое файла из корневого каталога, используя запись о файле в коревом каталоге и читая цепочку FAT. Файл занимает 2 кластера. О результате выполнения вывести сообщение на экран
6	Открыть файл. Установить длину записи в файле 1024 байта, выполнить вывод записи №4 в файл из буфера. Закрыть файл. О результате выполнения операции вывести сообщение на экран, используя атрибуты цвета. Вывести на экран размер выходного файла, использую запись о файле в каталоге

7	Удалить и восстановить файл на дискете, используя запись о нем в корневом каталоге и цепочку FAT. О результате выполнения операции вывести сообщение на экран.
8	Создать новый файл, записать в него информацию, введенную с клавиатуры. О результате выполнения операции вывести сообщение на экран. Вывести № первого кластера, распределенного файлу
9	Вывести на экран содержимое некоторых полей загрузочной записи дискеты. При выводе использовать атрибуты цвета
10	Вывести на экран корневой каталог диска. Причем, программы разного типа выводить разным цветом (например, .com – красным, .exe – белым, .txt – зеленым и т. д.)
11	Скопировать файл из одного каталога на дискете в другой каталог на этой же дискете. Вывести номер первого кластера обоих файлов
12	Скопировать файл из одного каталога на дискете в другой. Вывести на экран номер первого кластера обоих файлов
13	Вывести на экран информацию о том, какой диск и каталог являются текущими, а также информацию о размере, кластера, сектора и о размере свободного пространства на этом диске
14	Переименовать файл и добавить в него информацию. О результате выдать сообщение на экран, используя атрибуты цвета. Вывести информацию о размере файла в начале и в конце работы из записи о файле в каталоге
15	Вывести на экран содержимое текущего каталога. При этом программы разного типа выводить разным цветом
16	Вывести на экран оглавление текущего каталога диска A:, создать в этом каталоге файл и вновь вывесит оглавление, используя разные цветовые палитры
17	Вывести информацию о свободном и занятом пространстве на дискете, используя информацию о свободных кластерах в FAT
18	С помощью дескриптора файла, полученного из последней операции открытия, установить текущую позицию файла через 1024 байта от начала файла и вывести туда символьную строку. Вывести на экран информацию о размере файла из записи о файле в каталоге
19	Найти и прочитать запись о файле в каталоге. Вывести на экран всю информацию о файле, взятую из записи в каталоге
20	Прочитать в корневом каталоге запись о файле. Вывести на экран все номера кластеров, которые этот файл занимает
21	Создать файл, скопировать в него информацию из уже существующего файла. Удалить старый файл, а новый файл переименовать. Вывести на экран сообщение о результате. Из записи о файле в каталоге вывести на экран атрибуты файла
22	Создать новый файл. Скопировать в него информацию из уже существующего файла. О результате операции вывести сообщение на экран. Вывести номер первого кластера созданного каталога

23	Создать каталог на текущем диске в текущем каталоге. В созданном каталоге создать файл, записать в него символьную строку. Файл закрыть. Найти файл и вывести его содержимое на экран, используя атрибуты цвета. Вывести размер файла из записи в каталоге
24	Вывести на экран список удаленных файлов из текущего каталога, используя атрибуты цвета
25	Вывести на экран список удаленных файлов из корневого каталога
26	Вывести на экран список удаленных файлов из заданного каталога. При выводе использовать графический режим
27	Считать вектор с диска и записать его в файл. О результате операции вывести сообщение на экран, используя атрибуты цвета
28	Получить список кластеров, распределенных файлу
29	Вывести информацию о свободном месте на диске, используя информацию о свободных кластерах в FAT
30	Прочитать MBR и вывести на экран содержимое таблицы разделов со всеми атрибутами.
31	Прочитать содержимое корневого каталога и вывести список файлов с обработкой LFN
32	Прочитать содержимое корневого каталога и вывести список файлов с атрибутами и датой создания
33	Прочитать из MBR таблицу разделов и определить количество и размеры логических дисков

Библиографический список

1. Гордеев, А. В. Операционные системы: учеб. / А. В. Гордеев. – 2-е изд. – СПб.: Питер, 2004. – 416 с.

2. Гордеев, А. В. Системное программное обеспечение: учеб. / А. В.Гордеев, А. Ю. Молчанов. – СПб.: Питер, 2001, 2002. – 736 с.

3. Юров, В. И. Assembler: учеб. пособие / В. И. Юров. – 2-е изд. – СПб.: Питер, 2003. – 637 с.

4. Юров, В.И. Assembler: учеб. / В.И. Юров. – СПб.: Питер, 2001. – 624 с.: ил. + дискета.

5. Юров, В. И. Assembler: практикум / В. И. Юров. – СПб.: Питер, 2003. – 400 с.: ил. + CD.

6. Финогенов, К. Г. Основы языка Ассемблера: учеб. курс / К. Г. Финогенов. – М.: Радио и связь, 1999. – 288 с.: ил.

7. Финогенов, К. Г. Самоучитель по системным функциям MS-DOS / К. Г. Финогенов. – М.: Радио и связь: Энтроп, 1999. – 382 с.: ил.

Приложение

ФУНКЦИИ DOS ДЛЯ РАБОТЫ С КЛАВИАТУРОЙ

INT 21h, Функция 01h – ввод с клавиатуры с эхо.

На входе: AH – 01h.

На выходе: AL – байт входных данных.

Функция ожидает ввод со стандартного входного устройства. Если получен ASCII символ, то он выводится на стандартное выходное устройство и записывается в AL. Если после выполнения функции AL содержит 00H, это означает, что получен расширенный ASCII код, второй байт которого будет прочитан при следующем выполнении функции. При выполнении функции осуществляется проверка наличия Ctrl-Break.

INT 21h, Функция 06h – прямой ввод-вывод на консоль.

На входе: AH = 06h

Регистр AL используется для ввода, а регистр DL – для вывода. Если DL содержит FFH, то выполняется ввод и регистр AL готов принять входной символ. Состояние флага ZF показывает наличие символа. Если ZF содержит 1, то нет символа и значение AL несущественно. Если ZF содержит 0, то символ записывается в AL. Если значение DL отлично от FFH, выполняется вывод. DL содержит ASCII код символа. Функция 06H работает со стандартными входным и выходным устройствами. Функция не ожидает появления входного символа, читает без эхо и при ее выполнении не проверяется наличие Ctrl-Break.

INT 21h, Функция 07h – прямой ввод с клавиатуры без эхо.

На входе: AH = 07h.

На выходе: AL – ASCII-код вводимого символа.

Аналогична функции 01H, но работает без эхо и при ее выполнении не проверяется наличие Ctrl-Break.

INT 21h, Функция 08h – ввод с клавиатуры без эхо.

На входе: AH = 08h.

На выходе: AL – ASCII-код вводимого символа.

Аналогична функции 01h, но работает без эхо. При отсутствии символа ждет ввода. Для чтения расширенного кода ASCII требуется повторное выполнение функции.

INT 21h, Функция 0Ah – буферизованный ввод с клавиатуры.

На входе: AH = 0Ah;
DS:DX = адрес буфера.

На выходе: данные помещены в буфер в формате:
байт 0 – максимальное количество символов, которые могут быть обработаны;
байт 1 – фактическая длина введенной строки;
байт 2 и т.д. – строка, заканчивающаяся символом 0Dh.

Функция дает возможность использовать в программе стандартные клавиши для редактирования командной строки. Конечный результат представляет цепочку, которая получается в буфере ввода после нажатия клавиши Enter или после получения символа 0Dh от стандартного входного устройства. Максимальное количество символов, которое можно поместить в буфер, на единицу меньше значения его первого байта – одна позиция зарезервирована для символа 0Dh. При превышении этого максимума DOS игнорирует дальнейший ввод и выдает звуковой сигнал при вводе каждого последующего символа. При выполнении функции проверяется наличие Ctrl-Break.

INT 21h, Функция 0Bh – проверка входного статуса клавиатуры.

На входе: AH – 0Bh.

На выходе: FFh, если есть символ на входном устройстве;
00h, если нет символа.

При выполнении функции проверяется наличие Ctrl-Break.

INT 21h, Функция 0Ch – очистка входного буфера и ввод.

На входе: AH – 0Ch;
AL – номер функции ввода;
DS:DX – адрес буфера (если AL=0Ah).

На выходе: AL – байт входных данных (если при вызове AL=0Ah).

Стирает буфер клавиатурного драйвера BIOS, после чего выполняет указанную в регистре AL функцию. Допустимы функции 01h, 06h, 07h, 08h и 0Ah. При выполнении функции проверяется наличие Ctrl-Break.

ФУНКЦИИ BIOS ДЛЯ РАБОТЫ С КЛАВИАТУРОЙ

INT 16h, Функция 00h – чтение очередного символа с клавиатуры.

На входе: AH – 00h.

На выходе: AH – скэн-код;
AL – символ ASCII.

Читает информацию об очередном нажатом символе из буфера клавиатуры. Если буфер пуст, ожидается появления символа. В AL получается скэн-код символа, а в AH – позиционный код клавиши. Для клавиш и клавишных комбинаций, которые представляются расширенным кодом, AL равен 0, а AH содержит расширенный код клавиши. Если символ введен при помощи ALT и клавиш цифрового поля, то значение AH равно 0. Прочитанный символ удаляется из буфера.

INT 16h, Функция 01h – справка о наличии символа.

На входе: AH – 01h.

На выходе: ZF=0, если в буфере клавиатуры есть символ;
AH – скэн-код символа;
AL – ASCII-код символа;
ZF= 1, если буфер пуст.

Определяет, имеются ли в кольцевом буфере ожидающие ввода символы. Символ остается в буфере.

INT 16h, Функция 02h – информация о состоянии.

На входе: AH – 02h.

На выходе: AL – флаги.

Функция возвращает в регистр AL содержимое первого из двух байтов состояния клавиатуры с абсолютными адресами 417h и 418h. Если программа интересуется содержимым второго байта, то она должна прочитать его прямо, как адрес памяти.

ФУНКЦИИ DOS ДЛЯ ВЫВОДА ИНФОРМАЦИИ НА ЭКРАН

INT 21h, Функция 02h – вывод символов на экран через стандартный поток.

На входе: AH = 02h;
DL = ASCII-код выводимого символа.

Функция обрабатывает нажатие клавиш Ctrl-Break.

INT 21h, Функция 06h – прямой ввод-вывод на консоль.

На входе: AH = 06h.

INT 21h, Функция 09h – вывод строки на экран дисплея.

На входе: AH = 09h;
DS:DX = адрес отображаемой строки.

Строка отображается, начиная с текущей позиции курсора. Строка должна оканчиваться символом '$' (код ASCII 24h). Строка может включать управляющие символы. Функция обрабатывает нажатие клавиш Ctrl-Break.

ФУНКЦИИ BIOS ДЛЯ ВЫВОДА ИНФОРМАЦИИ НА ЭКРАН

INT 10h, Функция 00h – выбор режима работы.

На входе: AH – 00h;
AL – видеорежим.

Позволяет задать любой стандартный режим работы видеоадаптера. В регистр AL загружается номер устанавливаемого режима работы видеоадаптера, если бит 7=1, то при установке режима видеопамять не очищается.

INT 10h, Функция 01h – изменение формы курсора.

На входе: AH – 01h;
CH – верхняя граница курсора;
CL – нижняя граница курсора.

Позволяет изменить вертикальные размеры курсора. Горизонтальные размеры курсора всегда одинаковы и равны ширине одного символа. Курсор отображается только в текстовых режимах видеоадаптера.

INT 10h, Функция 02h – изменение положения курсора.

На входе: AH – 02h;
BH – номер страницы видеопамяти;
DH – номер строки;
DL – номер столбца.

Задает текущее положение курсора на экране дисплея. Если видеопамять разделена на несколько страниц, то каждая из них имеет свой курсор, координаты которого можно устанавливать отдельно.

INT 10h, Функция 03h – определение положения и формы курсора.

На входе: AH – 03h;
BH – номер страницы видеопамяти.
На выходе: CH – верхняя граница курсора;
CL – нижняя граница курсора;
DH – позиция текущей строки курсора;
DL – позиция текущего столбца курсора.

Позволяет узнать размер и текущие координаты курсора.

INT 10h, Функция 05h – изменение активной страницы видеопамяти.

На входе: AH – 05h;
AL – номер страницы видеопамяти, которая станет активной.

INT 10h, Функция 06h – перемещение текстового окна вверх.

На входе: AH – 06h;
AL - число строк для перемещения;
если AL=0, то окно очищается целиком.
BH - атрибут цвета для строк, возникающих снизу окна;
CL - строка и столбец верхнего левого угла окна;

DH и DL – строка и столбец нижнего правого угла окна.

Определяет прямоугольную область (окно) активной видеостраницы и ее содержимое перемещается вверх на одну или несколько строк. Освобожденные внизу строки заполняются пробелами с указанными атрибутами цвета, а верхние строки исчезают.

INT 10h, Функция 07h – перемещение текстового окна вниз.

На входе: AH – 07h;
AL – число строк для перемещения,
если AL=0, то окно очищается целиком;
BH – атрибут цвета для строк, возникающих снизу окна;
CH и CL – строка и столбец верхнего левого угла окна;
DH и DL – строка и столбец нижнего правого угла окна.

Подобна функции 06h, но пустые строки появляются в верхней части окна, а нижние строки исчезают.

INT 10h, Функция 08h – чтение символа и его атрибутов.

На входе: AH – 08h;
BH – номер страницы видеопамяти.
На выходе: AH – атрибут;
AL – символ.

Позволяет прочитать символ и его атрибуты из позиции экрана, определенной текущим положением курсора. Чтение можно производить как из активной, так и из неактивной страниц памяти.

INT 10h, функция 09h – запись символа с атрибутами в текущей позиции курсора.

На входе: AH – 09h;
AL – ASCII-код записываемого символа;
BH – номер страницы видеопамяти;
BL – атрибут для текстового режима или
цвет для графического режима;
CX – число записываемых символов.

Позволяет записать один или несколько одинаковых символов с атрибутами, при этом запись происходит как в активную, так и неактивные страницы видеопамяти. После выполнения записи положение курсора не изменяется. Управляющие символы, такие, как возврат каретки и перевод строки, не действуют и записываются как обычные символы. Функция работает в текстовых и в графических режимах.

INT 10h, функция 0Ah – запись символа в текущей позиции курсора.

На входе: AH – 0Ah;
AL – ASCII-код записываемого символа;
BH – номер страницы видеопамяти;
BL – цвет (для графического режима);
CX – число записываемых символов.

Можно производить запись нескольких одинаковых символов на любой странице видеопамяти. Атрибуты символов не задаются, а используются их старые значения из предыдущих операций записи. После операции записи положение курсора не изменяется. Управляющие символы не действуют и записываются как обычные символы.

INT 10h, функция 0Ch – вывод пикселя.

На входе: AH – 0Ch;
AL – номер цвета;
BH - номер страницы;
DL – номер строки от 0 до 199;
CX - номер столбца от 0 до 319 или до 639.

Используется в графических режимах для записи пикселя заданного цвета в любую страницу видеопамяти. Использование функций BIOS для создания изображений хотя и медленнее, чем прямая запись в видеопамять и программирование регистров, но более надежно с точки зрения совместимости для различных режимов и видеоадаптеров.

INT 10h, функция 0Dh – чтение пикселя.

На входе: AX – 0Dh;
BH – номер страницы;
CX – номер столбца;
DX – номер строки.

На выходе: AL – цвет (номер цветового регистра).

Позволяет определить цвет любого пикселя экрана по его координатам и используется только в графических режимах. Цвет пикселя получается в младших байтах регистра AL.

INT 10h, функция 0Eh – запись символа в режиме телетайпа.

На входе: AH – 0Eh;
AL – ASCII-код записываемого символа;
BH – номер страницы видеопамяти (только в текстовых режимах);
BL – цвет символа в графических режимах.

После отображения символа в текущей позиции курсора курсор затем сдвигается вправо на одну позицию. При необходимости курсор автоматически перемещается на новую строку. После заполнения экрана происходит сдвиг экрана вверх, причем верхняя строка исчезает, а внизу возникает новая пустая строка. Функция обрабатывает управляющие символы: звуковой сигнал, возврат курсора назад на одну позицию, возврат каретки и перевод строки. В текстовых режимах цвет символа не изменяется, а в графических есть возможность задать цвет отображаемого символа.

INT 10h, функция 0Fh – определение текущего режима работы видеоадаптера, номера активной страницы и количества символов в строке экрана.

На входе: AH – 0Fh.

На выходе: AH - число символов в строке;
AL – номер текущего режима;
BH – номер активной странице видеопамяти.

INT 10h, функция 13h – вывод текстовой строки.
На входе: AH – 13h;
AL – режим отображения строки:
0 – курсор не перемещается, атрибуты символов в строке одинаковы;
1 – курсор перемещается, атрибуты символов в строке одинаковы;
2 – курсор не перемещается, атрибуты каждого символа задаются отдельно;
3 – курсор перемещается, атрибуты каждого символа задаются отдельно;
BH – номер страницы видеопамяти;
BL – атрибут, если AL содержит 0 или 1;
CX – длина строки;
DH – номер строки экрана, в которой отображается строка;
ES:BP – адрес строки в оперативной памяти.

В зависимости от параметров функции можно определять атрибуты как строки в целом, так и отдельных символов строки. После вывода строки курсор может либо оставаться на месте, либо перемещаться в позицию за последним символом только что выведенной строки. В режимах 2 и 3 атрибуты каждого символа задаются вперемежку с самими символами строки. При этом сначала идет символ, а затем его атрибут.

ФУНКЦИИ DOS ДЛЯ РАБОТЫ С ФАЙЛАМИ, КАТАЛОГАМИ И ДИСКАМИ

INT 21h, Функция 0Dh – очищение дисковых буферов. Открытые файлы не закрываются.

INT 21h, Функция 0Eh – смена текущего диска.

На входе: DL – номер нового текущего диска.

На выходе: AL – максимальное количество логических дисков,как оно определено командой LASTDRIVE в CONFIG.SYS. По умолчанию LASTDRIVE=5 (максимум 5 логических дисков).

INT 21h, Функция 19h – получение текущего диска.

На входе: AH – 19h.

На выходе: AL – номер текущего диска.

INT 21h, Функция 1Ah – изменение адреса области обмена с диском.

На входе: AH – 19h;

DS:DX – адрес DTA (область обмена с диском).

При запуске программы адрес DTA указывает смещение 80H с начала PSP. При помощи этой функции DTA можно поместить в область пользовательской программы.

INT 21h, Функция 1Bh – информация о текущем диске.

На входе: AH – 1Bh.

На выходе: AL – количество секторов в кластере;

CX – количество байтов в секторе;

DX – общее количество кластеров на диске;

DS:BX – указатель к идентификатору формата диска:

FD – дискета 360 Кбайт;

F9 – дискета 1.2 Мбайт;

FB – дискета 1.44 Мбайт;

F8 – жесткий диск.

INT 21h, Функция 1Ch – информация о заданном диске.

На входе: AH – 1Ch;

DL – номер диска (0 – текущий, 1 – A и т.д.).

На выходе: AL – количество секторов в одном кластере;

DX – количество кластеров на диске;

CX – размер сектора;

DS:BX – указатель к идентификатору формата диска.

DS:BX содержит адрес одного байта в рабочих областях DOS, который является копией идентификатора формата диска из FAT. На самом деле это адрес начала копии FAT в рабочих областях DOS.

INT 21h, Функция 1Fh – получение адреса блока параметров текущего диска.

На входе: AH – 1Fh.
На выходе: AL – 00h – успешное выполнение;
DS:BX – адрес блока параметров диска.

INT 21h, Функция 2Eh – проверка при записи на диск.
На входе: AL – 01 активирует проверку при записи;
00 дезактивирует проверку при записи.

При активизации проверки после каждой операции записи на диск осуществляется контрольное чтение.

INT 21h Функция 2Fh – получение адреса области обмена с диском.
На входе: AH – 2Fh.
На выходе: ES:BX – адрес DTA.

INT 21h, Функция 32h – получение адреса блоков параметров заданного диска.
На входе: AH – 32h;
DL – номер диска.
На выходе: AL – 00h – успешное выполнение;
AL – FFh – недопустимый дисковод;
DS:BX – адрес блока параметров диска.

INT 21h, Функции 33h, подфункция 05h – получение дисковода загрузки.
На входе: AH – 33h;
AL – 05h.
На выходе: DL – дисковод загрузки (1=A:, и т.д.).

INT 21h Функция 36h – информация о свободном пространстве на диске.
На входе: AH – 36h;
DL – номер диска.
На выходе: BX – количество свободных кластеров;
DX – общее количество кластеров на диске;
CX – размер сектора (в байтах);
AX – количество секторов в одном кластере,
FFFFh, если номер диска в DL недействителен.

INT 21h, Функция 39h – создание каталога (МкDir).
INT 21h, Функция 3Ah – удаление каталога (RmDir).
INT 21h, Функция 3Bh – смена каталога (ChDir).
На входе: AH – номер функции;
DS:DX – указатель к ASCIIZ-цепочке.
На выходе: AX – код ошибки, если CF=1.

Функции аналогичны соответствующим командам DOS – MD, RD и CD. Нельзя удалить текущий каталог или каталог, содержащий файлы или каталоги.

INT 21h, Функция 3Ch – создание файла (CREATE).

На входе: AH – 3Ch;
DS:DX – указатель к ASCIIZ-цепочке;
CX (CL) – атрибуты файла.
На выходе: AX – код ошибки, если CF=1;
файловый дескриптор, если CF=0.

INT 21h, Функция 3Dh – открытие файла (OPEN).
На входе: AH – 3Dh;
DS:DX – указатель к ASCIIZ-цепочке;
AL – режим открытия (код доступа).
На выходе: AX – код ошибки, если CF=1;
файловый дескриптор, если CF=0.

Файл должен существовать. Не допускается указание группового имени. Указатель текущей позиции в файле устанавливается в начало файла.

В AL задаются режимы:

7	6	5	4	3	2	1	0	НАЗНАЧЕНИЕ
					Д	Д	Д	Режим доступа
				Р				Резервный
	П	П	П					Режим совместного использования
Н								Режим наследования

Режим доступа – биты 0 – 2. Определяют права доступа программы к файлу: (000) – только для чтения; (001) – только для записи; (010) – для чтения и записи. Другие комбинации не допускаются.

Режим наследования – бит 7. Указывает, как файл может быть использован порожденным процессом. Если бит 7 равен 0, то порожденный процесс наследует все открытые родительским процессом файлы в их текущем состоянии и с тем же режимом доступа. Если бит 7 равен 1, то порожденный процесс выполняется как независимая программа и может работать с файлом в соответствии с режимом совместного использования, заданным при его открытии родительским процессом.

Режим совместного использования – биты 4 – 6. Определяют право доступа к файлу для других процессов, которые пытаются открыть файл до его закрытия главным процессом, DOS следит за соблюдением режима совместного использования файлов, только если в памяти загружен специальный резидентный модуль. Один и тот же файл можно открыть многократно в одном и том же процессе. При каждом открытии создается новый блок управления и получается новый файловый дескриптор. Возможны следующие коды ошибок: 2 – файл не обнаружен; 3 – нет такого пути; 4 – нет свободного файлового дескриптора; 5 – отказан доступ; 12 – недействительный код доступа.

INT 21h, Функция 3Eh – закрытие файла (CLOSE).
На входе: AH – 3Eh.

DS:DX – файловый дескриптор (полученный при открытии или создании файла).

На выходе: AX – код ошибки, если CF=1.

Каталог изменяется. Буферы DOS, содержащие записи для этого файла, очищаются. Возможен код ошибки 6 – недействительный файловый дескриптор.

INT 21h Функция 3Fh – чтение из файла или устройства.
INT 21h Функция 40H – запись в файл или устройство.

На входе: AH – номер функции;
BX – файловый дескриптор, полученный при открытии или создании файла;
CX: – количество байтов (размер записи);
DS:DX – адрес, по которому нужно записать прочитанную запись.

На выходе: AX – код ошибки, если CF=1
или количество действительно прочитанных (записанных) байтов.

Функция читает/записывает указанное количество байтов с текущей позиции в файле (Fp). После операции значение Fp увеличивается на количество прочитанных/записанных байтов. Количество действительно прочитанных/записанных байтов не всегда равно указанному. Например, при достижении конца файла или при чтении с клавиатуры можно прочесть максимум одну строку (до нажатия Enter), независимо от указанного количества байтов для чтения. Коды ошибок: 5 – отказан доступ и 6 – недействительный файловый дескриптор.

INT 21h, Функция 41H – удаление файла.

На входе: AH – 41h;
DS:DX – указатель к ASCIIZ-цепочке.

На выходе: AX – код ошибки, если CF=1.

INT 21h, Функция 42h – перемещение текущего указателя в файле.

На входе: AH – 42h;
BX – файловый дескриптор, полученный при открытии или создании файла;
CX:DX – смещение относительно текущей позиции в байтах интерпретируется как 32-битовое целое число без знака;
AL – метод перемещения.

На выходе: AX – код ошибки, если CF=1;
DX:AX – новое значение указателя;
AL может иметь следующие значения:
0 – смещение считается от начала файла. Если CX:DX=0, то Fp указывает на начало файла;
1 – смещение считается от текущей позиции Fp;

2 – смещение считается от конца файла. Если CX:DX=0, то в DX:AX получается длина файла в байтах.

Возможны следующие коды ошибок: 1 – недействительный номерфункции; 6 – недействительный файловый дескриптор.

INT 21h Функция 43h – получение или изменение атрибутов файла.

На входе: AH – 43h;
DS:DX – указатель к ASCIIZ-цепочке;
CX – атрибуты файла;
AL – 00 – информация об атрибутах;
01 – изменение атрибутов.

На выходе: AX – код ошибки, если CF=1;
CX – атрибуты файла, если CF=0 и на входе AL=00h.

Атрибуты описаны в каталоге. Атрибуты 08H (метка тома) и 10H (каталог) нельзя изменять. Коды ошибок: 1 – недействительная функция; 2 – файл не обнаружен; 3 – нет такого пути; 5 – отказан доступ.

INT 21h, Функция 47h – информация о текущем каталоге.

На входе: AH – 47h;
DL – код дисковода;
DS:SI – адрес буфера длиной 64 байта.

На выходе: AX – код ошибки, если CF=1;
DS:SI – по этому адресу записывается полная файловая спецификация текущего каталога.

Имя дискового устройства не включается в полученную файловую спецификацию и файловая спецификация не содержит начальный символ "/", т.е. если текущим является корневой каталог, то возвращается пустая строка. Код ошибки 15 – недействительное имя устройства.

INT 21h Функция 4Eh – поиск первого совпадающего файла.

На входе: AH – 4Eh;
DS:DX – указатель к ASCIIZ-цепочке (файловая спецификация);
CX – атрибуты файла, используемые при поиске.

На выходе: AX – код ошибки, если CF=1.

Используя функцию 4Eh, можно искать конкретный файл или первый из группы файлов. Можно указать произвольные устройство и каталог. Допускаются символы '?' и '*'. При обнаружении файла DTA (байты 1Eh – 2Ah) заполняется следующей информацией:

21 байт – зарезервированы для последующего поиска функцией 4FH;
1 байт – атрибуты файла;
1 – только для чтения;
2 – скрытый;
4 – системный;
8 – метка тома;

10h – каталог;
20h – атрибут архивации;
2 байта – время (из каталога);
2 байта – дата (из каталога);
2 байта – размер файла (младшая часть);
2 байта – размер файла (старшая часть);
13 байт – имя обнаруженного файла.
Коды ошибок: 2 – файл не обнаружен; 18 – нет больше файлов.

INT 21h Функция 4Fh – поиск следующего совпадающего файла.
На входе: AH – номер функции;
DTA – информация из последней операции поиска.

Перед выполнением данной функции нужно успешно завершить функцию 4Eh или 4Fh. Необходимая для ее работы информация находится в DTA, и не следует выполнять операции, которые могут ее разрушить. Информация об обнаруженном файле записывается в DTA так же, как для функции 4Eh. Код ошибки 18 – нет больше файлов.

INT 21h Функция 56h – переименование файла.
На входе: AH – 56h;
DS:DX – указатель к ASCIIZ-цепочке – старое имя;
ES:DI – указатель к ASCIIZ-цепочке – новое имя.
На выходе: AX – код ошибки, если CF=1.

Имя устройства в старой и новой файловой спецификации должны совпадать. Пути могут быть различными – в этом случае файл меняет свой каталог. Групповые имена не допускаются.

INT 21h, Функция 5Ah – создание временного (уникального) файла.
INT 21h, Функция 5Bh – создание нового файла.
На входе: AH – номер функции;
DS:DX – указатель к ASCIIZ-цепочке;
CX (CL) – атрибуты файла.
На выходе: AX – код ошибки, если CF=1;
файловый дескриптор, если CF=0.

INT 21h, Функция 68h – Сбросить в файл.
На входе: BX – дескриптор.
На выходе: AX – код ошибки, если CF=1.

Принудительно переносит все данные из внутренних буферов системы, связанных с заданным дескриптором, на устройство. Если дескриптор относится к файлу, а файл модифицировался, то соответствующая запись каталога обновляется. Действие этой функции эквивалентно закрытию и повторному открытию файла. Функция будет выполняться и при недостаточном числе дескрипторов, и прикладная программа не рискует потерять управление файлом в многозадачных или сетевых средах.

INT 21h, Функция 69h – получение или установка серийного номера тома.

На входе: AH – 69h;
AL – 00h – получить серийный номер;
01h – установить серийный номер;
BL – дисковод;
DS:DX – адрес буфера размером 32 байта.

На выходе: при AL=00h в буфер помещается копия содержимого расширенного блока параметров BIOS (BpB) на диске;
при AL=01h в расширенный BpB на диске копируется информация из буфера;
AX – код ошибки, если CF=1.

INT 21h, Функция 6Ch – расширенное открытие файла.

На входе: AL – 00h;
BX – режим открытия;
CX – атрибут файла;
DX – флаг открытия;
DS:SI – адрес пути, заданного в коде ASCIIZ;

На выходе AX – дескриптор, если CF=0;
код ошибки, если CF=1;

CX - 1 – файл существовал и был открыт;
2 – файл не существовал и был создан;
3 – файл существовал и был заменен.

По данному в коде ASCIIZ пути открывает, создает и заменяет файл в указанном или действующем по умолчанию каталоге указанного или действующего по умолчанию диска.

Функция не выполняется, если
– не существует какого-либо элемента пути;
– если файл создается в корневом каталоге, который переполнен;
– если в заданном каталоге уже существует файл с тем же именем, что и создаваемый файл, и с атрибутом только для чтения;
– если программа выполняется в сети, а пользователь программы имеет недостаточные права доступа.

Данная функция объединяет возможности функций 3Ch, 3Dh и 5Bh.

INT 25h – абсолютное чтение диска. Обеспечивает непосредственную связь с модулем BIOS для чтения данных из логического сектора диска в память.

INT 26h – абсолютная запись на диск. Обеспечивает непосредственную связь с модулем BIOS для записи данных в логический сектор диска из памяти.

На входе: AL – номер дисковода (0=A, 1=B и т.д.)
CX – -1;
DS:BX – адрес блока параметров следующего формата:
00h-03h – 32-битный номер сектора;

04h-05h – число секторов для чтения;
06h-07h – относительный адрес буфера;
08h-09h – сегмент буфера.

На выходе: AX – код ошибки, если CF=1.

Содержимое всех регистров, за исключением сегментных, может быть изменено.

При возвращении из данного прерывания флаги ЦП, первоначально помещенные прерыванием 25h на стек, по-прежнему там и находятся. В целях предотвращения неуправляемого увеличения стека и обеспечения доступа ко всем другим величинам, которые были помещены на стек до вызова прерывания 25h, необходимо с помощью команды POPF или ADD Sp,2 очистить стек.

Код ошибки интерпретируется следующим образом.

Младший байт (AL) совпадает с кодом ошибки, возвращаемым в младший байт регистра DI, и может принимать следующие значения:

00h – ошибка защиты записи;
01h – неизвестное устройство;
02h – дисковод не готов;
03h – неизвестная команда;
04h – ошибка данных (CRC);
06h – ошибка позиционирования;
07h – неизвестный тип носителя;
08h – сектор не найден.

Старший байт (AH) содержит:

01h – неправильная команда;
02h – неправильная метка адреса;
03h – попытка записи на защищенный диск (26H);
04h – запрашиваемый сектор не найден;
08h – не выполнен прямой доступ к памяти;
10h – ошибка данных;
20H – не сработал контроллер;
40h – не выполнена операция установки;
80h – подсоединенное устройство не среагировало;

ФУНКЦИИ BIOS ДЛЯ РАБОТЫ С ДИСКАМИ

INT 13h, Функции 00h – сброс контроллера (Reset).

На входе: AH – 00h;
DL – дисковод.

Приводит в начальное состояние контроллер дискетных устройств. Это не отражается на дискете, находящейся в данный момент в устройстве. После сброса контроллер принимает, что головки всех устройств позиционированы на дорожке 0, что в общем случае неверно. Поэтому при выполнении операции BIOS обнуляет биты 0 – 3 байта по адресу 43Eh (бит 0 относится к устройству A, бит 1 – к устройству B и т.д.). При каждой операции для определенного устройства проверяется состояние соответствующего ему бита этого байта. Если он равен 0, то перед операцией головка позиционируется на дорожке 0 (рекалибровка), после чего выполняется операция и в соответствующий бит записывается 1. Сброс контроллера обычно осуществляется после ошибки при выполнении некоторых функций.

INT 13h, Функция 01h – получить состояние дисковой системы. Возвращает в AL информацию о последней выполненной операции. Такая информация записывается в один байт по адресу 441h после каждой операции с дискетой.

На входе: DL – дисковод: 00H – 07H – гибкий диск;
80H – FFH – жесткий диск;
AL = 00h – операция выполнена успешно;
01h – недействительная команда;
02h – не обнаружено или ошибочно адресное поле;
03h – попытка записи на защищенную дискету;
04h – сектор не обнаружен;
05h – сброс в исходное состояние не выполнен (Ж);
06h – гибкий диск снят (Г);
07h – дефектная таблица параметров (Ж);
08h – нарушение границ ПДП (Г);
09h – ПДП (DMA) пересек границу 64 Кбайт;
0Ah – флаг дефектного сектора (Ж);
0Bh – флаг дефектной дорожки (Ж);
0Ch – не найден тип носителя (Г);
0Dh – недопустимое число секторов при форматировании (Ж);
0Fh – обнаружена метка адреса управляющих данных;
10h – невосстановимая ошибка циклического контроля или ошибка кода проверки и корректировки данных по избыточности (CRC);
11h – ошибка данных скорректирована кодом проверки и корректировки (Ж) (обнаружение восстановимой

ошибки в процессе предшествующего чтения сектора);
20h – отказ контроллера;
40h – сбой поиска;
80h – нет ответа от устройства (тайм-аут);
AAh – дисковод не готов (Ж);
BBh – неопределенная ошибка (Ж);
CCh – отказ записи (Ж);
E0h – ошибка регистра состояния (Ж);
FFh – операция опознания не выполнилась (Ж);

INT 13h, Функция 02h – чтение секторов.
Функция 03h – запись секторов.
Функция 04h – проверка секторов: могут ли быть обнаружены и прочитаны секторы и проверка данных в секторах.

На входе: AH – номер функции;
DL – номер устройства (0–3);
DH – сторона дискеты (0–1);
CH – номер дорожки;
CL – номер первого сектора;
AL – число секторов для чтения (записи);
ES:BX – адрес входного (выходного) буфера (не используется функцией 04H).

При использовании жестких дисков старшие два бита 10-битного номера цилиндра помещаются в старшие два бита регистра CL. Секторы должны быть заранее форматированы.

INT 13h, Функция 05h – форматирование дорожки.

На входе: AH – 05h;
DL – номер устройства (0–3);
DH – сторона дискеты (0–1);
CH – цилиндр;
AL – число секторов для форматирования;
ES:BX – адрес буфера.

Дорожка форматируется целиком без возможности форматирования отдельного сектора. Однако каждый сектор может иметь разные характеристики. Для каждого сектора готовятся 4 байта, которые его описывают – так называемое адресное поле. Пара регистров ES:BX должна указывать область, в которой последовательно записаны адресные поля будущих секторов. Четыре байта адресного поля имеют обозначения:

C – цилиндр;
H – головка (номер поверхности);
R – запись (номер сектора);
N – код размера сектора (без адресного поля).

При форматировании адресные поля тоже записываются на дорожку в начале соответствующего сектора. Когда нужно записать или прочитать сектор, сначала осуществляется позиционирование головки на требуемую поверхность и дорожку. Затем на дорожке осуществляется поиск сектора, номер которого совпадает со значением в CL. Байты C и H используются только для проверки. Число байтов, которое нужно прочитать или записать, определяется байтом N. При форматировании байту N можно задавать значения 0, 1, 2 и 3, которые обозначают соответственно 128, 256, 512 и 1024 байта. DOS всегда использует размер сектора 512 байтов.

INT 13h, Функция 10h – получить состояние дисковода.

На входе: AH – 10h;
DL: 80H – FFH жесткий диск.

На выходе: AH – 0, если SF=0;
AH – код ошибки, если CF=1 (см. функцию 01H).

Проверяет, является ли указанный дисковод жесткого диска работоспособным, и возвращает его состояние.

INT 13h, Функция 15h – получить тип дисковода.

На входе: AH – 15h;
DL – дисковод.

На выходе: AH – код типа дисковода:
00h – дисковод отсутствует;
01h – используется дисковод гибкого диска без поддержки сигнала смены носителя;
02h – используется дисковод гибкого диска с поддержкой сигнала смены носителя;
03h – используется жесткий диск, в этом случае:
CX:DX – число 512-байтный секторов;
AH – код ошибки, если CF=1 (см. функцию 01H);

Возвращает код указанного дисковода, соответствующий типу гибкого или жесткого диска.

Учебное издание

Караваева Ольга Владимировна

Системные программные средства
для управления вводом-выводом.
Лабораторный практикум

Учебное пособие