БИБЛИОТЕКА СИСТЕМНОГО ПРОГРАММИСТА

А. В. ФРОЛОВ, Г. В. ФРОЛОВ

АППАРАТНОЕ ОБЕСПЕЧЕНИЕ IBM PC

ТОМ **2**

ЧАСТЬ 1

ИЗДАНИЕ ВТОРОЕ, СТЕРЕОТИПНОЕ

МОСКВА • "ДИАЛОГ-МИФИ" • 1992

ББК 32.973.1
Ф91

БИБЛИОТЕКА СИСТЕМНОГО ПРОГРАММИСТА
Выпускается с 1991 года

Фролов А.В., Фролов Г.В.
Ф91 Аппаратное обеспечение IBM PC: В 2-х ч. Ч. 1.—2-е изд., стер.—М.: "ДИАЛОГ-МИФИ", 1992.—208 с.—(Библиотека системного программиста; Т. 2)
ISBN 5-86404-024-X (Т. 2, ч. 1)

Учебно-справочное пособие по использованию драйверов и портов ввода/вывода различных устройств компьютера и составлению эффективных программ, использующих все особенности аппаратуры.

В первой части подробно описаны клавиатура, мышь, таймер, часы реального времени, асинхронный адаптер, порт параллельной передачи данных. Для описанных устройств приводится методика программирования на всех уровнях - от использования портов ввода/вывода до высокоуровневых средств стандартных библиотек трансляторов Microsoft QuickC 2.5 и C 6.0. Книга содержит большое количество примеров, составленных на языках Ассемблера и C. Дополнительно можно приобрести дискеты с примерами программ.

Г $\frac{2404090000\text{-}009}{\text{Г70(03)-92}}$ Без объявл. ББК 32.973.1

Учебно-справочное издание

Фролов Александр Вячеславович
Фролов Григорий Вячеславович
АППАРАТНОЕ ОБЕСПЕЧЕНИЕ IBM PC
В двух частях. Часть 1

Редактор: О.А. Красильникова
Макет: О.А. Красильникова, О.А. Кузьминова
Обложка: Н.В. Дмитриева
Корректор: В.С. Кустов

Подписано в печать 10.01.93. Формат 60x84/16. Печать офсетная.
Усл. печ. л. 12,09. Уч.-изд. л. 9. Доп. тираж. Заказ 1291.

Акционерное общество "ДИАЛОГ-МИФИ"
115409 Москва, ул. Москворечье, 31, корп. 2

Подольский филиал Чеховского полиграфического комбината.
142100, г. Подольск, Московская обл., ул. Кирова, 25

ISBN 5-86404-024-X (Т. 2, ч. 1)
ISBN 5-86404-004-5

ВВЕДЕНИЕ

Программисты, использующие операционную систему MS-DOS, часто вынуждены работать с различными устройствами компьютера на уровне команд ввода/вывода. Это связано прежде всего с тем, что MS-DOS не содержит сколько-нибудь существенной поддержки для большинства устройств компьютера. Практически функционально полная поддержка обеспечивается только для дисковой подсистемы. Такие устройства, как мышь, принтер, расширенная и дополнительная память и часы реального времени, либо обслуживаются отдельными драйверами, либо программа вынуждена обращаться непосредственно к портам ввода/вывода этих устройств.

Для оптимального решения графических и вычислительных задач актуально использование арифметического сопроцессора 8087/80287/80387. Мы рассмотрим сопроцессор с точки зрения программиста - опишем форматы используемых данных, рассмотрим внутренние регистры сопроцессора и систему команд.

Данная книга содержит информацию об использовании драйверов и портов ввода/вывода некоторых устройств компьютера и поможет Вам составлять эффективно работающие программы, использующие все особенности аппаратуры.

Описаны следующие устройства:

- клавиатура;
- мышь;
- часы реального времени;
- таймер;
- порт последовательной передачи данных (асинхронный адаптер);
- порт параллельной передачи данных и принтер;
- контроллер прямого доступа к памяти;
- контроллер прерываний;
- расширенная и дополнительная память;
- арифметический сопроцессор.

Глава 1 описывает основные способы определения конфигурации персонального компьютера. В книге 3 первого тома мы уже

занимались определением конфигурации дисковой подсистемы. В этом томе будет рассказано о способах определения конфигурации других подсистем и, в частности, о способе определения типа центрального процессора.

Глава 2 посвящена клавиатуре. В ней рассказывается о том, как работает клавиатура, подробно описываются различные способы программирования клавиатуры - от использования портов ввода/вывода до средств операционной системы MS-DOS и стандартных библиотек трансляторов Microsoft QC 2.5 и C 6.0.

Глава 3 рассказывает об устройстве, принципе работы и программировании мыши - одного из наиболее распространенных устройств ввода для персонального компьютера. Приводятся многочисленные примеры программ для работы с мышью.

В главе 4 описаны часы реального времени, получающие питание от аккумулятора. Пользуясь сведениями, приведенными в этой главе, Вы сможете не только определять или устанавливать дату и время, но и запускать периодические процессы, запускать процессы в определенные моменты времени. Последнее возможно при использовании режима "будильник".

Глава 5 расскажет Вам о системном таймере. Вы научитесь программировать это устройство, играть с его помощью простейшие музыкальные мелодии, узнаете, как реализовать с помощью таймера генератор случайных чисел. Будет рассказано о способах формирования задержек в работе программы, длительность которых не зависит от производительности центрального процессора.

В главе 6 описан порт последовательной передачи данных и приведены все необходимые сведения для его использования (включая разводку разъемов).

В главе 7 описан порт параллельной передачи данных. Так как к этому порту обычно подключается принтер, то мы приведем сведения об использовании и программировании принтеров (матричных). Иногда порт параллельной передачи данных используют для управления каким-либо оборудованием, например аналого-цифровым преобразователем. Пользуясь приведенными сведениями, Вы сможете приспособить свой компьютер для управления такими внешними устройствами, не приобретая специальные интерфейсные адаптеры. В приложении приведена таблица

команд для наиболее распространенных 9-иголочных и 24-иголочных принтеров Epson.

Глава 8 содержит сведения о программировании канала прямого доступа к памяти. Прямой доступ к памяти используется для организации быстрого ввода/вывода данных и всегда применяется для организации работы с дисками.

Глава 9 посвящена программированию контроллера прерываний. Известно, что обслуживание медленно работающих устройств ввода/вывода целесообразно выполнять с использованием механизма прерываний. Приведенная в главе 9 информация позволит Вам самостоятельно программировать контроллер прерываний и эффективно использовать прерывания в своих программах.

В главах 10 и 11 мы расскажем Вам об использовании расширенной и дополнительной (Extended и Expanded) памяти компьютера. Вы сможете преодолеть барьер 640 K, установленный для MS-DOS реальным режимом работы процессора.

Глава 12 посвящена арифметическому сопроцессору. В настоящее время это устройство имеется практически в любом компьютере (за исключением, возможно, самых дешевых конфигураций IBM PC/XT). Арифметический сопроцессор значительно ускоряет выполнение расчетных и графических задач. Некоторые программные продукты (например, Autocad версий 10.0 и более поздних версий) просто отказываются работать на компьютере, не оснащенном сопроцессором.

Глава 1

КОНФИГУРАЦИЯ ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА

В настоящее время широко используются персональные компьютеры самых разных типов (IBM PC, IBM XT, IBM AT, PS/2, Compaq-386 и т. д.). Если Вы создаете коммерческое программное обеспечение, следует позаботиться о том, чтобы оно работало на всех типах компьютеров, использующих процессоры серии Intel 8086/80286/80386/80486. Для обеспечения такой совместимости программа должна уметь определять тип используемого центрального процессора и, разумеется, тип используемого компьютера. Почему это так важно?

Корпорация Intel, разрабатывая свою серию процессоров 8086/80286/80386/80486, уделила много внимания обеспечению совместимости снизу вверх. Такая совместимость гарантирует возможность непосредственного выполнения старшими моделями процессоров любых программ, подготовленных для младших моделей. При этом не требуется повторной компиляции или редактирования этих программ.

Однако совместимость снизу вверх не гарантирует (разумеется!) обратного - возможности выполнения младшими моделями программ, предназначенных для старших моделей. Поэтому, если Вы желаете добиться от Вашей программы максимальной эффективности и быстродействия, имеет смысл использовать архитектурные особенности старших моделей процессоров. Однако это не означает, что Ваша программа должна всегда максимально использовать все возможности процессора 80486, так как в этом случае она не сможет правильно работать на широко распространенном компьютере IBM AT, использующем процессор 80286.

Лучше всего было бы сделать так, чтобы программа динамически (в процессе своей работы) определяла тип используемого процессора и вызывала соответствующие модули, рассчитанные на применение конкретной модели процессора. Вы можете выделить эти модули в самостоятельные программные единицы (драйверы или оверлеи) и подгружать их в оперативную память при необхо-

димости. Такой подход позволит Вам в будущем легко реализовать возможности процессоров 80586 или 80986 - Вам будет достаточно изготовить новый драйвер (или оверлей) и подключить его к уже готовой программе.

Другая проблема связана с большим разнообразием "не вполне" совместимых с IBM PC/XT/AT компьютеров, выпускающихся разными "третьими" фирмами. К этой категории можно также отнести и отечественную ППЭВМ ЕС-1841 - это вроде бы IBM XT, но не вполне, поэтому некоторые программы на этой машине работать не будут.

К счастью, фирмы - производители оборудования записывают в определенные ячейки ПЗУ BIOS некоторый код, по которому можно определить тип компьютера. Можно очень легко отличить IBM XT от IBM AT.

По типу компьютера программа может сделать предварительные, а в некоторых случаях и окончательные выводы о наличии того или иного оборудования. Например, IBM PC или IBM XT не содержат расширенной памяти или дисковода для работы с гибкими дисками диаметром 3,5 дюйма. Разные модели, использующие процессоры 80286/80386/80486, могут использовать различные способы управления двадцатой адресной линией (она используется при работе с расширенной памятью, подробности - в 10-й главе). Кроме того, от модели компьютера зависит способ, которым программа может определить конфигурацию дисковой подсистемы. Об этом мы говорили в третьей книге 1 тома "Библиотеки системного программиста", посвященной дисковой подсистеме.

Прежде чем использовать какие-либо аппаратные ресурсы компьютера, программа должна убедиться в том, что эти ресурсы имеются в составе системы. Попытка обращения программы к несуществующему устройству может привести, например, к зависанию операционной системы.

Как программа может определить конфигурацию подсистем компьютера?

Для машин класса IBM PC и IBM XT конфигурация задается установкой перемычек на материнской плате и платах контроллеров периферийных устройств. Программа может получить информацию об установленных перемычках, прочитав состояние определенных портов компьютера.

Мы уже говорили о том, что в машинах класса IBM AT, IBM PS/2 и машинах более высокого класса установлена КМОП-память - память с малым энергопотреблением. Эта память питается от аккумуляторов и содержит информацию о конфигурации многих подсистем (в том числе дисковой подсистемы).

Во время инициализации системы BIOS опрашивает порты, к которым подключены перемычки и ячейки КМОП-памяти, содержащие информацию о конфигурации компьютера. Результат записывается в область данных BIOS - в слово конфигурации с адресом 0000:0410.

BIOS также предоставляет программам некоторые средства для определения конфигурации компьютера. В частности, с помощью прерывания INT 11h программа может получить в регистре AX слово конфигурации из области данных BIOS.

1.1. Определение типа компьютера и версии BIOS

У Вас есть две возможности определить модель компьютера и получить некоторую информацию о конфигурации - прочитать эту информацию из ячеек ПЗУ BIOS или вызвать одну из функций прерывания INT 15h, возвращающую адрес таблицы конфигурации.

ПЗУ BIOS содержит по адресу FFFF:FFFE байт, значение которого можно использовать для идентификации типа компьютера:

FF - оригинальный IBM PC;
FE - XT, Portable PC;
FD - PCjr;
FC - AT;
FB - XT с памятью 640 K на материнской плате;
FA - PS/2 модель 25 или 30;
F9 - Convertible PC;
F8 - PS/2 модели 55SX, 70, 80;
9A - Compaq XT, Compaq Plus;
30 - Sperry PC;
2D - Compaq PC, Compaq Deskpro.

Для определения модели компьютера таким способом мы предлагаем следующую функцию:

```
/**
*.Name       pc_model
*
*.Title      Определить модель компьютера
*
*.Descr      Функция возвращает байт, идентифицирующий
*            модель персонального компьютера
*
*.Params     Нет
*
*.Return     Код модели персонального компьютера:
*
*      FF     оригинальный IBM PC;
*      FE     XT, Portable PC;
*      FD     PCjr;
*      FC     AT;
*      FB     XT с памятью 640 K на материнской плате;
*      FA     PS/2 модель 25 или 30;
*      F9     Convertible PC;
*      F8     PS/2 модели 55SX, 70, 80;
*      9A     Compaq XT, Compaq Plus;
*      30     Sperry PC;
*      2D     Compaq PC, Compaq Deskpro.
*
**/

#include <stdio.h>
#include <dos.h>
#include "sysp.h"

char unsigned pc_model(void) {

     char unsigned _far *modptr;

     modptr = FP_MAKE(0xf000,0xfffe);

     return *modptr;
}
```

Функция pc_model() возвращает байт, идентифицирующий код компьютера. В большинстве случаев Вам достаточно проверить этот байт и сделать вывод о типе компьютера и составе его аппаратных средств.

Более подробную информацию можно получить, вызвав функцию C0h прерывания BIOS INT 15h:

На входе:	AH	=	C0h
На выходе:	ES:BX	=	адрес таблицы конфигурации, таблица находится в ПЗУ BIOS;
	CF	=	0 при успешном вызове прерывания;
	CF	=	1 если данная версия BIOS не поддерживает функцию C0h.

После выполнения прерывания регистры ES:BX будут указывать на таблицу в области ПЗУ BIOS. В этой таблице имеется более точная информация о типе компьютера, номер версии BIOS, сведения об аппаратных особенностях конкретной модели.

Приведем формат указанной таблицы:

Смещение	*Размер*	*Описание*
(+0)	2	Размер таблицы в байтах
(+2)	1	Код модели
(+3)	1	Дополнительный код модели
(+4)	1	Версия BIOS revision: 0 для первой реализации; 2 - для второй и т. д.
(+5)	1	Байт конфигурации оборудования:
		бит 7 = канал 3 контроллера прямого доступа к памяти используется дисковой системой базового ввода-вывода (дисковой BIOS);
		бит 6 = установлен второй контроллер прерываний 8259;
		бит 5 = установлены часы реального времени;
		бит 4 = каждый раз после вызова прерывания от клавиатуры INT 9h вызывается функция 4Fh прерывания INT 15h;
		бит 3 = BIOS поддерживает ожидание внешнего события;
		бит 2 = используется расширенная область данных BIOS;
		бит 1 = если этот бит установлен в 1, то используется шина Micro Channel, в противном случае - ISA
		бит 0 зарезервирован
(+6)	2	Зарезервировано и равно 0
(+8)	2	Зарезервировано и равно 0

В следующей таблице приведены коды моделей, дополнительные коды моделей и версии BIOS для некоторых широко распространенных типов компьютеров:

Код модели	*Доп.код модели*		*Версия BIOS*	*Тип компьютера*
FFh	*	*	04/24/81	Оригинальная версия IBM PC
FFh	*	*	10/19/81	IBM PC, в этой версии BIOS исправлены некоторые ошибки
FFh	*	*	10/27/82	IBM PC, используется накопитель на магнитном диске (НМД), оперативная память 640 K, поддерживается адаптер дисплея EGA
FEh	*	*	08/16/82	IBM PC XT
FEh	*	*	11/08/82	IBM PC XT, Portable
FDh	*	*	06/01/83	PCjr
FCh	*	*	01/10/84	IBM AT, модели 068, 099, частота тактового генератора 6 MHz, емкость НМД - 20MB
FCh	00h	01h	06/10/85	IBM AT, модель 239, частота тактового генератора 6 MHz, емкость НМД - 30MB
FCh	01h	00h	11/15/85	IBM AT, модели 319, 339, частота тактового генератора 8 MHz, используются расширенная клавиатура, BIOS может работать с накопителями на гибких магнитных дисках формата 3,5 дюйма
FCh	01h	-		Compaq 286/386
FCh	02h	00h	04/21/86	IBM PC XT-286
FCh	04h	00h	02/13/87	PS/2 модель 50
FCh	04h	03h	04/18/88	PS/2 модель 50Z
FCh	05h	00h	02/13/87	PS/2 модель 60
FCh	06h	-		7552 "Gearbox"
FCh	09h	02h	06/28/89	PS/2 модель 30-286

FCh	81h	00h	01/15/88	Phoenix 386 BIOS, версия 1.10
FBh	00h	01h	01/10/86	IBM PC XT, расширенная клавиатура, BIOS может работать с накопителями на гибких магнитных дисках формата 3,5 дюйма
FBh	00h	02h	05/09/86	IBM PC XT
FAh	00h	00h	09/02/86	PS/2 модель 30
FAh	00h	01h	12/12/86	PS/2 модель 30
FAh	01h	00h	-	PS/2 модель 25
F9h	00h	00h	09/13/85	PC Convertible
F8h	00h	00h	03/30/87	PS/2 модель 80 16MHz
F8h	01h	00h	10/07/87	PS/2 модель 80 20MHz
F8h	04h	02h	04/11/88	PS/2 модель 70
F8h	04h	03h	03/17/89	PS/2 модель 70
F8h	09h	-	-	PS/2 модель 70
F8h	09h	02h	04/11/88	PS/2 модель 70
F8h	09h	03h	03/17/89	PS/2 модель 70
F8h	0Ch	00h	11/02/88	PS/2 модель 55SX
F8h	1Bh	00h	10/02/89	PS/2 модель 70-486
9Ah	*	*	-	Compaq XT или Compaq Plus
30h	-	-	-	Sperry PC
2Dh	*	*	-	Compaq PC или Compaq Deskpro

Следует заметить, что функция C0h прерывания INT 15h поддерживается не всеми версиями BIOS, а только теми, которые были изготовлены после 10 января 1986 года. Если Вы используете более старые версии BIOS, дополнительный код модели, версия BIOS и байт конфигурации Вам недоступны.

Кроме того, BIOS, изготовленный 10 января 1986 года и установленный в IBM XT, возвращает неправильное значение байта конфигурации.

Символ '*' в таблице означает, что функция C0h прерывания INT 15h для данной версии BIOS не реализована. Все, что Вы мо-

жете сделать в этом случае для идентификации BIOS, - получить байт кода модели по адресу F000h:FFFEh и дату изготовления BIOS, занимающую 8 байтов начиная с адреса F000h:FFF5h. Дата хранится в формате ASCII.

Приведем текст программы, которая поможет Вам определить версию BIOS и дату ее изготовления, а также получить всю остальную информацию из таблицы конфигурации. Программа отображает также адрес этой таблицы.

```
#include <stdio.h>
#include <dos.h>
#include "sysp.h"

void main(void);

void main(void) {

     union REGS  rg;
     struct SREGS srg;
     int   i;
     BIOSINFO far *biosinf_ptr;

          printf("\n*BIOSTEST* Информация о BIOS, "
               "(C) Фролов А., 1991");

// Конструируем указатель на дату изготовления
// BIOS. Эта дата записана в ПЗУ по адресу F000h:FFF5h.

          biosinf_ptr = FP_MAKE(0xf000, 0xfff5);

// Выводим дату на экран

          printf("\n\nДата изготовления BIOS:     ");
          for(i=0; i<8; i++)
               putch(*((char far *)biosinf_ptr + i));

// Вызываем функцию C0h для получения адреса
// таблицы конфигурации компьютера.

          rg.h.ah = 0xc0;
          int86x(0x15, &rg, &rg, &srg);

// Если данная функция не поддерживается BIOS,
// читаем код модели компьютера из ПЗУ
// по адресу F000h:FFFEh.

          if(rg.x.cflag == 1) {
               printf("\nФункция C0h прерывания INT 15h "
                    "данной версией BIOS не поддерживается\n");

// Конструируем указатель на код модели

               biosinf_ptr = FP_MAKE(0xf000, 0xfffe);
```

```
// Выводим код модели компьютера на экран

            printf("\nКод модели:            %02.2X",
                 (unsigned char)(*(char far *)biosinf_ptr));

            exit(-1);
        }

// Конструируем укзатель на таблицу конфигурации

        biosinf_ptr = FP_MAKE(srg.es, rg.x.bx);

// Выводим на экран содержимое таблицы

        printf("\nАдрес таблицы конфигурации: %Fp"
                  "\nРазмер таблицы в байтах:    %d"
                  "\nКод модели:                 %02.2X"
                  "\nДополнительный код модели:  %d"
                  "\nВерсия BIOS:                %d"
                  "\nКонфигурация оборудования:  %02.2X",
                  biosinf_ptr,
                  biosinf_ptr->size,
                  biosinf_ptr->model,
                  biosinf_ptr->submodel,
                  biosinf_ptr->version,
                  biosinf_ptr->hardcfg);

// Определяем конфигурацию компьютера

        printf("\n\nКонфигурация оборудования компьютера"
                 "\n-----------------------------------");

// Запоминаем байт конфигурации

        i = biosinf_ptr->hardcfg;

// Расшифровываем байт конфигурации

        if(i & 0x80)
             printf("\nКанал 3 контроллера DMA используется"
                           " дисковой BIOS");

        if(i & 0x40)
             printf("\nУстановлен второй контроллер"
                           " прерываний 8259");

        if(i & 0x20)
             printf("\nУстановлены часы реального времени");

        if(i & 0x10)
             printf("\nПосле INT 9h вызывается функция 4Fh"
                           " прерывания INT 15h");

        if(i & 0x8)
             printf("\nBIOS поддерживает функцию ожидания"
                           " внешнего события");
```

```
        if(i & 0x4)
             printf("\nИспользуется расширенная"
                          " область данных BIOS");

        if(i & 0x2)
             printf("\nИспользуется шина Micro Channel");

        if(!(i & 0x2))
             printf("\nИспользуется шина ISA");

        exit(0);
}
```

На экране, после запуска этой программы на компьютере Datamini 386 Tower, мы получили:

```
02/25/89
*BIOSTEST* Информация о BIOS, (C)
Фролов А., 1991

Дата изготовления BIOS:
Адрес таблицы конфигурации: F000:E6F5
Размер таблицы в байтах:    8
Код модели:                 FC
Дополнительный код модели:  1
Версия BIOS:                0
Конфигурация оборудования:  70

Конфигурация оборудования компьютера

Установлен второй контроллер прерываний 8259
Установлены часы реального времени
После INT 9h вызывается функция 4Fh прерывания INT 15h
Используется шина ISA
```

Для изменения конфигурации компьютера (добавления новых устройств или удаления устаревших) может потребоваться изменение установки перемычек на материнской плате компьютера и/или на плате контроллера устанавливаемого устройства. Поэтому мы расскажем об установке перемычек на материнской плате.

1.2. Установка перемычек на материнской плате

Для правильной установки перемычек, находящихся на материнской плате и платах контроллеров периферийных устройств, Вам необходима соответствующая документация. Это руководства пользователя, поставляющиеся вместе с компьютером или контроллерами.

Обозначения перемычек и их назначение различны не только для разных типов компьютеров (PC, XT, AT, PS/2), но и для машин разных фирм - производителей оборудования. Поэтому здесь не приводятся конкретные сведения об установке перемычек для оборудования, имеющегося в распоряжении авторов - у Вас может оказаться другая модель компьютера.

Вместо этого мы приведем общие сведения об установке перемычек. Пользуясь фирменной документацией, Вы найдете нужные перемычки на материнской плате Вашего компьютера и сможете их правильно установить.

Оригинальный компьютер IBM PC в его первоначальном виде в настоящее время используется крайне редко, поэтому мы не будем рассказывать о перемычках для IBM PC.

Машина IBM XT имеет один банк перемычек, определяющих количество установленных накопителей на гибких магнитных дисках (НГМД), тип используемого дисплейного адаптера, объем оперативной памяти, установленной на материнской плате, использование арифметического сопроцессора 8087.

Как правило, если Вы не меняете конфигурацию системы, у Вас не возникает необходимость в изменении установки перемычек. Если же Вы добавляете еще один НГМД, изменяете тип дисплейного адаптера или устанавливаете (снимаете) арифметический сопроцессор, Вам необходимо переустановить соответствующие перемычки на материнской плате.

Приведем таблицу перемычек банка SW1 для IBM XT (используя эту таблицу, обязательно сверьте ее с приведенной в документации на материнскую плату Вашего компьютера):

Номер перемычки	*Назначение и установка*
1	Зацикливание процедуры POST (тест после включения питания). Эта перемычка должна находиться в состоянии OFF
2	Наличие арифметического сопроцессора 8087: 8087 установлен - OFF; 8087 не установлен - ON

3-4	Объем оперативной памяти, установленной на материнской плате компьютера:		
	3	4	
	OFF	ON -	128 K;
	ON	OFF -	192 K;
	OFF	OFF -	256 K
5-6	Тип дисплейного контроллера:		
	5	6	
	ON	ON -	не подключен или EGA;
	OFF	ON -	CGA в режиме 40x25;
	ON	OFF -	CGA в режиме 80x25;
	OFF	OFF -	MDA или два контроллера: MDA и CGA
7-8	Количество установленных НГМД:		
	7	8	
	ON	ON -	установлен 1 НГМД;
	OFF	ON -	установлено 2 НГМД;
	ON	OFF -	установлено 3 НГМД;
	OFF	OFF -	установлено 4 НГМД

Если Вы изменяете объем оперативной памяти, установленной на материнской плате, Вам не надо переустанавливать перемычки 3 и 4. Это связано с тем, что BIOS во время инициализации системы после включения питания сам определяет объем установленной памяти - он сканирует всю имеющуюся память. Объем проверенной памяти обычно отображается во время теста.

Конфигурация компьютера IBM AT определяется в основном не установкой перемычек, а содержимым энергонезависимой КМОП-памяти. Мы уже говорили об этом при обсуждении конфигурации дисковой подсистемы.

Для работы с КМОП-памятью BIOS содержит специальную программу, называемую SETUP-программой. Аналогичная программа содержится на диагностической дискете, поставляемой вместе с компьютером.

После перезагрузки или включения питания BIOS обычно предоставляет возможность запустить программу SETUP. Для этого, как правило, надо нажать клавишу DEL во время инициализации

системы. Однако некоторые старые версии BIOS для AT не предоставляют возможности запуска программы SETUP при перезапуске системы. При использовании этих версий SETUP запускается сам, если в процессе тестирования обнаружились неисправности в оборудовании. Например, Вы отключили накопитель на жестком магнитном диске (НМД) и включили компьютер. Процедуры тестирования, запускаемые в процессе инициализации, обнаружат неисправность в НМД и предоставят Вам возможность работать с программой SETUP, находящейся в BIOS.

Другая возможность - загрузиться с диагностической дискеты и запустить программу SETUP, находящуюся на этой дискете.

Перемычка, влияющая на конфигурацию компьютера IBM AT, - SW1. Она определяет тип дисплейного контроллера, используемого программой SETUP. Состояние этой перемычки анализируется при разрушении содержимого КМОП-памяти (например, при разряде аккумулятора, питающего КМОП-память и часы реального времени).

Если перемычка SW1 установлена в положение OFF, используется контроллер CGA, в противном случае - MDA, EGA, VGA. Обычно Вам не требуется переустанавливать эту перемычку.

1.3. КМОП-память и конфигурация компьютера

Назначение некоторых ячеек КМОП-памяти мы рассматривали в разделе, посвященному конфигурации дисковой подсистемы. В этом разделе мы расскажем о назначении остальных ячеек.

В КМОП-памяти хранится текущее время и дата, сведения о конфигурации системы, результат тестирования при включении питания и информация, приведенная в следующей таблице:

Адрес ячейки	*Содержимое*
00h - 0Dh	Используются часами реального времени
0Eh	Байт состояния диагностики при включении питания
0Fh	Байт состояния отключения
10h	Тип используемого НГМД
11h	Зарезервировано

12h	Тип НМД (если тип меньше 15)
13h	Зарезервировано
14h	Конфигурация оборудования
15h - 16h	Объем основной памяти
17h - 18h	Объем расширенной (extended) памяти
19h	Тип первого НМД (если тип > 15)
1Ah	Тип второго НМД (если тип > 15)
1Bh - 20h	Зарезервировано
21h - 2Dh	Зарезервировано
2Eh - 2Fh	Контрольная сумма ячеек 10h - 20h
30h - 31h	Объем расширенной (extended) памяти
32h	Текущее столетие в двоично-десятичном коде (19h для 19-го столетия)
33h	Различная информация
34h - 3Fh	Зарезервировано

Рассмотрим назначение отдельных ячеек КМОП-памяти.

00h - 0Dh - область часов реального времени

Ячейки с адресами 00h - 0Dh используются часами реального времени. Часам реального времени будет посвящена отдельная глава, поэтому сейчас мы не станем останавливаться на этих ячейках.

0Eh - байт состояния диагностики

Байт состояния диагностики (расположенный в КМОП-памяти по адресу 0Eh) содержит результаты выполнения диагностики при включении питания компьютера. Выполнив анализ содержимого байта 0Eh, программа может выявить неисправность НМД, часов реального времени, разрядку аккумулятора и ошибки в конфигурации. Приведем формат этого байта:

Бит	*Значение*
0-1	Не используется, равно 0
2	0 - неправильная установка часов реального времени; 1 - часы реального времени установлены правильно

3	1 - неисправность НМД, невозможно загрузить операционную систему с жесткого диска;
	0 - НМД исправен
4	1 - фактический размер оперативной памяти не соответствует указанному в КМОП-памяти;
	0 - размер оперативной памяти указан правильно
5	1 - ошибка в конфигурации системы, фактическая конфигурация не соответствует указанной в байте конфигурации оборудования (адрес 14h);
	0 - конфигурация указана правильно
6	1 - ошибка в контрольной сумме КМОП-памяти;
	0 - контрольная сумма КМОП-памяти правильная
7	1 - разрядка аккумулятора, питающего КМОП-память и часы реального времени;
	0 - аккумулятор исправен и заряжен

0Fh - байт состояния отключения

Байт состояния отключения 0Fh используется процессорами 80286, 80386 и 80486 для определения способа возврата из защищенного режима в реальный после аппаратного сброса.

Вы, вероятно, знаете, что эти процессоры могут работать либо в реальном режиме, который соответствует режиму работы процессора 8086, либо в защищенном. Защищенный режим работы используется такими операционными системами, как OS/2, UNIX, XENIX, а также операционными оболочками WINDOWS/386 и WINDOWS версии 3.0. В этом режиме процессор может непосредственно адресовать всю память, лежащую выше границы 1 мегабайт.

Подробное рассмотрение защищенного режима работы выходит за рамки данной книги. Расскажем кратко о переходе из реального режима в защищенный и обратно для иллюстрации использования ячейки КМОП-памяти с адресом 0Fh.

Для перевода процессора 80286 из реального режима в защищенный можно использовать специальную команду LMSW:

```
mov  ax,1
lmsw ax
```

Разумеется, двух строк, приведенных выше, недостаточно для правильной работы процессора в защищенном режиме.

Для того, чтобы вернуть процессор 80286 из защищенного режима в реальный, необходимо выполнить аппаратный сброс (отключение) процессора. Это можно сделать следующим образом:

```
mov  ax, 0FEh          ; команда отключения
out  64h, ax
```

Перед выдачей команды отключения программа должна записать в ячейку 0Fh КМОП-памяти причину отключения:

Значение	*Причина отключения*
0	Программный сброс при нажатии комбинации клавиш CTRL-ALT-DEL или неожиданный сброс. Выполняется обычный перезапуск системы, но процедуры тестирования при включении питания не выполняются
1	Сброс после определения объема памяти
2	Сброс после тестирования памяти
3	Сброс после обнаружения ошибки в памяти (контроль четности)
4	Сброс с запросом перезагрузки
5	После сброса перезапускается контроллер прерываний, затем управление передается по адресу, который находится в области данных BIOS 0000:0467h
6,7,8	Сброс после выполнения теста работы процессора в защищенном режиме
9	Сброс после выполнения пересылки блока памяти из основной памяти в расширенную
0Ah	После сброса управление немедленно передается по адресу, взятому из области данных BIOS 0000:0467h

Для перевода процессоров 80386 и 80486 из реального режима в защищенный и обратно можно использовать загрузку специаль-

ного управляющего регистра CR0 обычной командой MOV. Однако будет работать и метод, основанный на применении команды LMSW и команды отключения.

Вы можете использовать сведения о команде отключения для организации программного перезапуска системы.

10h - тип используемых флоппи-дисков;

Младшая и старшая тетрады этого байта описывают соответственно второй и первый НГМД:

0000 - дисковод не установлен;
0001 - дисковод на 360K;
0010 - дисковод на 1,2M;
0011 - дисковод на 720K;
0100 - дисковод на 1.44M.

11h - зарезервировано для AT, тип НМД для PS/2

В компьютерах PS/2 ячейки КМОП-памяти с адресами 11h и 12h используются для хранения типов соответственно первого и второго НМД.

12h - типы первого и второго НМД

Этот байт разделен на две тетрады аналогично байту, описывающему НГМД. Однако в тетраде можно закодировать только 16 различных значений, а типов НМД значительно больше. Поэтому тип 15 используется специальным образом - если тип НМД в младшей тетраде (диск C:) равен 15, то правильное значение типа находится в КМОП-памяти по адресу 19h. Аналогично для диска D: - этот тип можно взять из байта по адресу 1Ah (если старшая тетрада байта с адресом 12h равна 15).

Таблица используемых типов дисков была приведена в третьей книге первого тома, в разделе, посвященном конфигурации дисковой подсистемы. Кроме того, сведения о типах дисков, задаваемых программой SETUP, обычно приводятся в документации, поставляемой вместе с компьютером.

13h - зарезервировано

Эта ячейка КМОП-памяти зарезервирована для дальнейшего развития системы.

14h - конфигурация оборудования

В этом байте находится информация о количестве установленных НГМД, о наличии арифметического сопроцессора 80287 или 80387 и о типе используемого дисплейного контроллера. Приведем формат байта конфигурации:

Бит	*Значение*
0	1 - в системе установлены НГМД; 0 - НГМД не используются
1	1 - установлен арифметический сопроцессор 80287 или 80387; 0 - арифметический сопроцессор не установлен
2-3	Не используются, равны 0
4-5	Тип дисплейного контроллера и его режим: Биты: 5 4 0 0 - не используется или EGA; 0 1 - CGA, EGA, VGA в режиме 40x25; 1 0 - CGA, EGA, VGA в режиме 80x25; 1 1 - монохромный контроллер
6-7	количество используемых НГМД: Биты: 7 6 0 0 - установлен 1 НГМД 0 1 - установлен 2 НГМД 1 0 - установлен 3 НГМД 1 1 - установлен 4 НГМД

15h-16h - объем основной памяти

Ячейка 15h содержит младший байт, а ячейка 16h - старший байт объема основной памяти. Например:

```
0100h - 256K
0200h - 512K
0280h - 640K
```

17h-18h - объем дополнительной памяти

Ячейки 17h и 18h содержат соответственно младший и старший байты размера дополнительной памяти (расположенной выше границы 1 М) в килобайтах.

19h-1Ah типы первого и второго НМД

Эти ячейки содержат типы соответственно первого и второго НМД, если соответствующий тип имеет значение, большее 15 (см. описание ячейки 12h).

1Bh-2Dh - зарезервировано

Эти ячейки КМОП-памяти зарезервированы для дальнейшего развития системы.

2Eh-2Fh - контрольная сумма ячеек 10h - 20h

Для ячеек КМОП-памяти с адресами от 10h до 20h при инициализации системы BIOS выполняет проверку контрольной суммы. Эта контрольная сумма хранится также в КМОП-памяти в ячейках 2Eh и 2Fh (соответственно старший и младший байты).

30h-31h - объем дополнительной памяти

Ячейки 30h и 31h содержат соответственно младший и старший байты размера дополнительной памяти (расположенной выше границы 1 М) в килобайтах.

Эта информация дублирует аналогичную информацию, расположенную в ячейках с адресами 17h-18h.

32h Текущее столетие

В машинах IBM AT этот байт содержит текущее столетие в двоично-десятичном коде, т. е. 19-е столетие записано как 19h.

PS/2 использует эту ячейку вместе с ячейкой 33h для хранения контрольной суммы ячеек с адресами от 10h до 31h. При этом старший байт контрольной суммы хранится в ячейке 32h, а младший - 33h.

33h - различная информация

Для IBM AT этот байт используется программой SETUP.

34h-3Fh - зарезервировано

Это поле Вы можете использовать по своему усмотрению, например, хранить здесь пароль.

PS/2 использует ячейку с адресом 37h для хранения номера текущего столетия. Ячейки 38h - 3Fh в модели 50 компьютера PS/2 используются для хранения пароля. Обращение к этим ячейкам выполняется по адресам 78h - 7Fh, которые аппаратно отображаются на адреса 38h - 3Fh.

Приведем две маленькие программы, демонстрирующие приемы работы с КМОП-памятью. Первая программа записывает в ячейки 34h - 3Fh строку символов, вторая отображает эту строку, а также некоторые другие ячейки.

```
#include <stdio.h>
#include <stdlib.h>

main() {

// Эта строка будет записана в КМОП-память

  static char password[12] = "!Frolov A.V.";
  int i,j;

  printf("\n*Запись в CMOS* (C)Фролов А. 1991\n\n");

  for(i=0x34,j=0; i<0x40; i++,j++) {

// Задаем адрес ячейки КМОП-памяти

    outp(0x70,i);

// Выполняем запись в эту ячейку

    outp(0x71,password[j]);
  }
}
```

Программа для чтения содержимого КМОП-памяти:

```
#include <stdio.h>
#include <stdlib.h>

main() {
  unsigned char cmos[164];
  int i;

  printf("\n*Чтение из CMOS* (C)Фролов А. 1991\n\n");

// Читаем все 64 ячейки КМОП-памяти в массив cmos
```

```
  for(i=0; i<64; i++) {
    outp(0x70,i);
    cmos[i]=inp(0x71);
  }

// Отображаем ячейки часов реального времени

  printf("\nЯчейки часов реального времени: ");

  for(i=0; i<0xd; i++) {
     printf("%02.2x ",(unsigned)cmos[i]);
  }

// Отображаем состояние байта диагностики
// после включения питания

  printf("\nБайт диагностики: %02.2x",cmos[0xe]);

// Отображаем содержимое байта отключения

  printf("\nБайт отключения: %02.2x\n",cmos[0xf]);

// Отображаем содержимое зарезервированных ячеек

  printf("\nPassword : ");
  for(i=0x34; i<0x40; i++) {
     printf("%02.2x ",(unsigned)cmos[i]);
  }

// Выводим это же еще раз в виде текстовой строки

  cmos[0x40]=0;
  printf(">%s<\n",&cmos[0x34]);
}
```

Некоторую помощь в определении конфигурации компьютера Вам может оказать прерывание BIOS INT 11h, которое мы рассмотрим ниже.

1.4. Использование BIOS для определения конфигурации

Как мы уже говорили, BIOS в процессе инициализации опрашивает состояние перемычек и анализирует содержимое КМОП-памяти (на тех машинах, где она установлена). После анализа BIOS записывает в свою область данных по адресу 0000h:0410h слово конфигурации. Отдельные биты этого слова содержат информацию о наличии в системе различного оборудования. Это слово можно получить с помощью прерывания INT 11h, которое возв-

ращает его в регистре AX. Приведем назначение отдельных битов слова конфигурации:

Биты	*Значение*
0	1 - система содержит НМД; 0 - система не содержит НМД
1	1 - установлен арифметический сопроцессор; 0 - арифметический сопроцессор не установлен
2-3	Объем основной памяти, установленной на материнской плате: Биты: 3 2 0 1 - 16K; 1 0 - 32K; 1 1 - 64K и более
4-5	Тип дисплейного контроллера и его режим: Биты: 5 4 0 0 - не используется или EGA; 0 1 - CGA, EGA, VGA в режиме 40x25; 1 0 - CGA, EGA, VGA в режиме 80x25; 1 1 - монохромный контроллер
6-7	Количество установленных НГМД: Биты: 7 6 0 0 - установлен 1 НГМД; 0 1 - установлено 2 НГМД; 1 0 - установлено 3 НГМД; 1 1 - установлено 4 НГМД
8	1 - используется контроллер прямого доступа к памяти; 0 - контроллер прямого доступа к памяти не используется
9-11	Количество установленных портов последовательной передачи данных RS232S: 000 - нет портов; 001 - используется один порт; . . . . . . 111 - используется 7 портов

12	1 - используется игровой адаптер (джойстик); 0 - игровой адаптер не используется
13	1 - установлен последовательный принтер (только для PC Jr)
14-15	Количество установленных принтеров: 00 - нет принтеров; 01 - используется 1 принтер; 10 - используется 2 принтера; 11 - используется 3 принтера

1.5. Определение типа процессора

Мы уже говорили о том, для чего может потребоваться программе определять тип используемого процессора.

Для определения типа процессора можно использовать следующую программу:

```
                .MODEL  tiny
                .STACK  100h

                .DATA

msg1        db "Тип Вашего процессора: ", "$"
m_8086      db "8086","$"
m_80286     db "80286","$"
m_80386     db "80386","$"

                .CODE
                .STARTUP

                mov     ah, 9h
                mov     dx, OFFSET msg1
                int     21h

; Записываем 0 в регистр флагов

                xor     ax,ax
                push    ax
                popf

; Переписываем регистр флагов через
; стек в регистр AX

                pushf
                pop     ax
```

```
; Проверяем установку старших четырех битов

            and     ax,0F000h
            cmp     ax,0F000h

; Если эти биты установлены, программа
; работает на процессоре 8086

            je      short CPU_8086

; Записываем 0F000h в регистр флагов

            mov     ax,0F000h
            push    ax
            popf

; Переписываем регистр флагов через
; стек в регистр AX

            pushf
            pop     ax

; Проверяем установку старших четырех битов

            and     ax,0F000h

; Если эти биты установлены, программа
; работает на процессоре 80286

            jz      short CPU_80286

; Если биты не установлены, то программа
; работает на процессоре 80386

            mov     dx, OFFSET m_80386
            jmp end_program

CPU_80286:
            mov     dx, OFFSET m_80286
            jmp end_program

CPU_8086:
            mov     dx, OFFSET m_8086
end_program:

            mov     ah, 9h
            int     21h

            .EXIT   0

            END
```

Работа программы основывается на записи в стек слова состояния процессора и последующего извлечения его. При этом проверяется содержимое старших извлеченных из стека битов.

Глава 2

КЛАВИАТУРА

В этом разделе мы подробно рассмотрим одно из важнейших устройств персонального компьютера - клавиатуру. Практически ни одна программа не обходится без обращения к клавиатуре.

Программа может использовать клавиатуру по-разному. Она может задержать свое выполнение до тех пор, пока оператор не введет какое-нибудь число или пока не нажмет какую-нибудь клавишу. Выполняя некоторую работу, программа может периодически проверять, не нажал ли оператор на клавишу, изменяющую режим работы программы. Резидентные программы могут контролировать все нажатия на клавиши, активизируясь при нажатии определенной заранее комбинации. Можно использовать прерывание, вырабатываемое клавиатурой, например для завершения работы программы.

Мы расскажем о работе с клавиатурой на разных уровнях - от использования клавиатурных портов ввода/вывода до средств, предоставляемых стандартными библиотеками трансляторов Microsoft QC 2.5 и C 6.0. Какой уровень Вам следует выбрать, зависит от решаемой задачи. Единственное, что можно порекомендовать, - это использовать по возможности средства высокого уровня. Если Ваша программа работает с клавиатурой на уровне портов ввода/вывода, ее работа может оказаться зависимой от типа клавиатуры и от типа компьютера.

2.1. Принципы работы клавиатуры

Клавиатура выполнена, как правило, в виде отдельного устройства, подключаемого к компьютеру кабелем. Малогабаритные компьютеры Lap-Top и Notebook имеют встроенную клавиатуру.

Что же находится внутри клавиатуры? Оказывается, там есть компьютер! Только этот компьютер состоит из одной микросхемы и выполняет специализированные функции. Он отслеживает нажатия на клавиши и посылает номер нажатой клавиши в центральный компьютер.

Если рассмотреть сильно упрощенную принципиальную схему клавиатуры, представленную на рисунке, можно заметить, что все клавиши находятся в узлах матрицы:

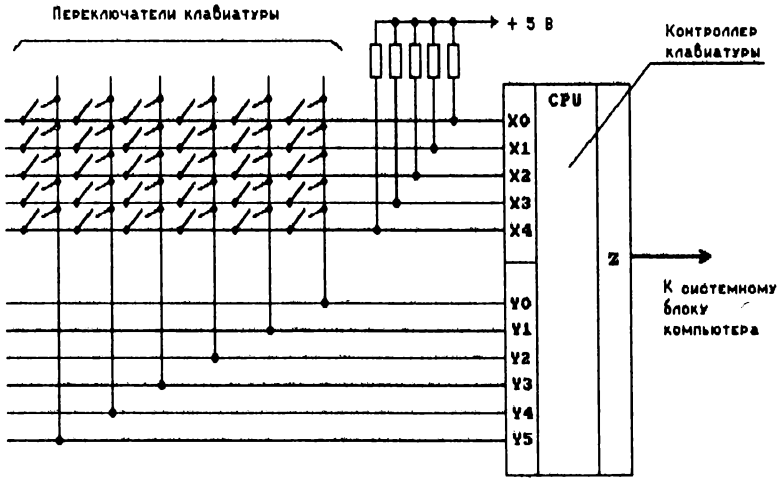


Рис. 1. Упрощенная схема клавиатуры

Все горизонтальные линии матрицы подключены через резисторы к источнику питания +5 В. Клавиатурный компьютер имеет два порта - входной и выходной. Входной порт подключен к горизонтальным линиям матрицы (X0-X4), а выходной - к вертикальным (Y0-Y5).

Устанавливая по очереди на каждой из вертикальных линий уровень напряжения, соответствующий логическому нулю, клавиатурный компьютер опрашивает состояние горизонтальных линий. Если нажатых клавиш нет, уровень напряжения на всех горизонтальных линиях соответствует логической 1 (т. к. все эти линии подключены к источнику питания +5 В через резисторы).

Если оператор нажмет на какую-либо клавишу, то соответствующие вертикальная и горизонтальная линии окажутся замкнутыми. Когда на этой вертикальной линии процессор установит значение логического нуля, то уровень напряжения на горизонтальной линии также будет соответствовать логическому нулю.

Как только на одной из горизонтальных линий появится уровень логического нуля, клавиатурный процессор фиксирует нажатие на клавишу. Он посылает в центральный компьютер запрос на прерывание и номер клавиши в матрице. Аналогичные действия выполняются, когда оператор отпускает нажатую ранее клавишу.

Номер клавиши, посылаемый клавиатурным процессором, однозначно связан с распайкой клавиатурной матрицы и не зависит напрямую от обозначений, нанесенных на поверхность клавиш. Этот номер называется скан-кодом (Scan Code).

Слово scan ("сканирование") подчеркивает, что клавиатурный компьютер сканирует клавиатуру для поиска нажатой клавиши.

Но программе нужен не порядковый номер нажатой клавиши, а соответствующий обозначению на этой клавише ASCII-код. Этот код не зависит однозначно от скан-кода, т. к. одной и той же клавише могут соответствовать несколько значений ASCII-кода. Это зависит от состояния других клавиш. Например, клавиша с обозначением '1' используется еще и для ввода символа '!' (если она нажата вместе с клавишей SHIFT).

Поэтому все преобразования скан-кода в ASCII-код выполняются программным обеспечением. Как правило, эти преобразования выполняют модули BIOS. Для использования символов кириллицы эти модули расширяются клавиатурными драйверами.

Если нажать на клавишу и не отпускать ее, клавиатура перейдет в режим автоповтора. В этом режиме в центральный компьютер автоматически через некоторый период времени, называемый периодом автоповтора, посылается код нажатой клавиши. Режим автоповтора облегчает ввод с клавиатуры большого количества одинаковых символов.

Следует отметить, что клавиатура содержит внутренний 16-байтовый буфер, через который она осуществляет обмен данными с компьютером.

В настоящее время существует три различных типа клавиатуры. Это клавиатура для компьютеров IBM PC/XT, 84-клавишная клавиатура для IBM AT и 101-клавишная (расширенная) клавиатура для IBM AT. Некоторые клавиатуры имеют переключатель режима работы (XT/AT), расположенный на нижней крышке. Он должен быть установлен в правильное положение.

2.2. Порты для работы с клавиатурой

Для работы с клавиатурой типа PC/XT используются порты с адресами 60h и 61h. Порт 60h при чтении содержит скан-код последней нажатой клавиши.

Порт 61h управляет не только клавиатурой, но и другими устройствами компьютера, например работой встроенного динамика. Этот порт доступен как для чтения, так и для записи. Для нас важен самый старший бит этого порта. Если в старший бит порта 61h записать значение 1, клавиатура будет заблокирована, если 0 - разблокирована.

Так как порт 61h управляет не только клавиатурой, при изменении содержимого старшего бита необходимо сохранить состояние остальных битов этого порта. Для этого можно сначала выполнить чтение содержимого порта в регистр, изменить состояние старшего бита, затем выполнить запись нового значения в порт:

```
in      al, 61h
or      al, 80h
out      61h, al
   .........
```

Компьютер типа IBM AT позволяет управлять скоростными характеристиками клавиатуры, а также зажигать или гасить светодиоды на лицевой панели клавиатуры - Scroll Lock, Num Lock, Caps Lock.

Для расширенного управления клавиатурой используется порт 60h в режиме записи. Этот порт используются для управления подчиненным процессором Intel 8042, ответственным за обмен данными с клавиатурным компьютером.

При использовании порта 60h на запись программа дополнительно получает следующие возможности:

- установка времени ожидания перед переходом клавиатуры в режим автоповтора;
- установка периода генерации скан-кода в режиме автоповтора;
- управление светодиодами, расположенными на лицевой панели клавиатуры - Scroll Lock, Num Lock, Caps Lock.

Процессор 8042 обслуживает не только клавиатуру, но и другие системы компьютера. Через порт 64h, например, выполняется сброс (отключение) процессора 80286 для возврата из защищенного режима работы в реальный.

Для посылки команды процессору 8042 вначале необходимо убедиться в том, что его внутренняя очередь команд пуста. Это можно сделать, прочитав слово состояния 8042 из порта с адресом 64h. Бит с номером 1 должен быть равен нулю.

Приведем фрагмент программы, составленной на языке ассемблера, проверяющий состояние очереди команд процессора 8042:

```
        mov     cx,0            ; счетчик для ограничения времени
                                ; ожидания готовности 8042
wait_loop:
; читаем порт состояния процессора 8042
        in      al,64h
        and     al,00000010b    ; флаг готовности
; ожидаем готовность процессора 8042
        loopnz  wait_loop
                                ; к приему команды
```

После того как программа дождется готовности процессора 8042, она может послать ему команду, записав ее в порт с адресом 60h:

```
        mov     al,cmd          ; команда для 8042
        out     60h ,al          ; вывод команды в 8042
          . . . . . .
```

Некоторые команды состоят более чем из одного байта. Остальные байты команды необходимо записать в порт 60h, предварительно убедившись в готовности процессора 8042 с помощью последовательности команд, приведенной выше. В большинстве случаев можно использовать простую временную задержку:

```
          . . . . . .
    mov     al, cmd_byte1
    out     60h, al

    mov     cx, 2000h
```

```
wait_loop: loop wait_loop

     mov     al, cmd_byte2
     out     60h, al
          ..........
```

Мы приведем формат двух команд процессора 8042, имеющих отношение к работе с клавиатурой - команду установки задержки и периода автоповтора и команду управления светодиодами, расположенными на клавиатуре.

Для установки характеристик режима автоповтора в порт 60h необходимо записать код команды 0F3h, затем байт, определяющий характеристики режима:

Биты	*Значение*
0-4	Период автоповтора: 0 - 30.0; 0Ah - 10.0; 1 - 26.7; 0Dh - 9.2; 2 - 24.0; 10h - 7.5; 4 - 20.0; 14h - 5.0; 8 - 15.0; 1Fh - 2.0. Период автоповтора определяет количество посылок скан-кода, генерируемых процессором клавиатуры в одну секунду. Можно использовать не только те значения, которые приведены выше, но и промежуточные, например 9 или 15h
5-6	Задержка включения режима автоповтора: 00 - 250 мс; 01 - 500 мс; 10 - 750 мс; 11 - 1000 мс
7	Зарезервировано, должно быть равно 0

Первоначально при инициализации системы период задержки для включения режима автоповтора устанавливается модулями BIOS равным 500 мс при периоде автоповтора, равном 10 повторам в секунду. Если это слишком медленно для Вас, Вы можете установить другие значения. Некоторые прикладные программы,

например текстовый процессор Microsoft Word, содержат средства для управления временными характеристиками клавиатуры.

Для управления светодиодами, расположенными на лицевой панели клавиатуры, используйте команду 0EDh. Вслед за этой командой в порт 60h необходимо записать байт, имеющий следующий формат:

Биты	*Значение*
0	1 - включить светодиод Scroll Lock
1	1 - включить светодиод Num Lock
2	1 - включить светодиод Caps Lock
3-7	Не используются

Приведем пример простейшей программы, управляющей светодиодами на лицевой панели компьютера. Такое управление может выполняться только при использовании порта 60h управления клавиатурой, т. к. BIOS не содержит соответствующей поддержки. Наша программа после запуска включит все светодиоды и будет ожидать нажатия на любую клавишу. После нажатия программа выключит светодиоды.

```
#include <stdio.h>

void main(void);

void main(void) {

     int i;

// Посылаем процессору клавиатуры
// команду управления светодиодами

     outp(0x60,0xed);

// Перед посылкой второго байта команды
// выполняем небольшую задержку

     for(i=0; i<4000; i++);

// Выводим второй байт команды,
// младшие три бита которого определяют
// состояние светодиодов на лицевой панели
// клавиатуры.

     outp(0x60,7);
```

```
// Ожидаем нажатия на любую клавишу.
     getch();
// Выключаем все светодиоды.
     outp(0x60,0xed);
     for(i=0; i<4000; i++);
     outp(0x60,0);
     exit(0);
}
```

2.3. Аппаратное прерывание клавиатуры

Клавиатура подключена к линии прерывания IRQ1. Этой линии соответствует прерывание INT 09h.

Клавиатурное прерывание обслуживается модулями BIOS. Драйверы клавиатуры и резидентные программы могут организовывать дополнительную обработку прерывания INT 09h. Для этого может быть использована цепочка обработчиков прерывания. В первой книге первого тома мы приводили примеры расширения обработчика прерывания INT 09h.

Как работает стандартный обработчик клавиатурного прерывания, входящий в состав BIOS?

Этот обработчик выполняет следующие действия:

- читает из порта 60h скан-код нажатой клавиши;
- записывает вычисленное по скан-коду значение ASCII-кода нажатой клавиши в специальный буфер клавиатуры, расположенный в области данных BIOS;
- устанавливает в 1 бит 7 порта 61h, разрешая дальнейшую работу клавиатуры;
- возвращает этот бит в исходное состояние;
- записывает в порт 20h значение 20h для правильного завершения обработки аппаратного прерывания.

Обработчик прерывания INT 09h не просто записывает значение ASCII-кода в буфер клавиатуры. Дополнительно отслеживаются нажатия таких комбинаций клавиш, как Ctrl-Alt-Del, обрабатываются специальные клавиши PrtSc и SysReq. При вычислении кода ASCII нажатой клавиши учитывается состояние клавиш Shift и CapsLock.

Буфер клавиатуры имеет длину 32 байта и расположен по адресу 0000h:041Eh для машин IBM PC/XT.

В IBM AT и PS/2 расположение клавиатурного буфера задается содержимым двух слов памяти с адресами 0000h:0480h (компонента смещения адреса начала буфера) и 0000h:0482h (смещение конца буфера). Обычно в IBM AT эти ячейки памяти содержат значения соответственно 001Eh и 003Eh. Так как смещения заданы относительно сегментного адреса 0040h, то видно, что обычное расположение клавиатурного буфера в IBM AT и PS/2 соответствует его расположению в IBM PC/XT.

Клавиатурный буфер организован циклически. Это означает, что при его переполнении самые старые значения будут потеряны. Две ячейки памяти, находящиеся в области данных BIOS с адресами 0000h:041Ah и 0000h:041Ch, содержат соответственно указатели на начало и конец буфера. Если значения этих указателей равны друг другу, буфер пуст (можно удалить все символы из буфера клавиатуры, установив оба указателя на начало буфера. Однако есть более предпочтительный способ с использованием прерывания BIOS INT 16h).

Указателями на начало и конец клавиатурного буфера обычно управляют обработчики прерываний INT 09h и INT 16h.

Программа извлекает из буфера коды нажатых клавиш, используя различные функции прерывания INT 16h.

Помимо управления содержимым буфера клавиатуры, обработчик прерывания INT 09h отслеживает нажатия на так называемые переключающие клавиши - NumLock, ScrollLock, CapsLock, Ins. Состояние этих клавиш записывается в область данных BIOS в два байта с адресами 0000h:0417h и 0000h:0418h.

Формат байта 0000h:0417h:

Биты	*Значение*
0	Нажата правая клавиша Shift
1	Нажата левая клавиша Shift
2	Нажата комбинация клавиш Ctrl-Shift с любой стороны
3	Нажата комбинация клавиш Alt-Shift с любой стороны
4	Состояние клавиши ScrollLock

5	Состояние клавиши NumLock
6	Состояние клавиши CapsLock
7	Состояние клавиши Insert

Формат байта 0000h:0418h:

Биты	*Значение*
0	Нажата левая клавиша Shift вместе с клавишей Ctrl
1	Нажата левая клавиша Shift вместе с клавишей Alt
2	Нажата клавиша SysReq
3	Состояние клавиши Pause
4	Нажата клавиша ScrollLock
5	Нажата клавиша NumLock
6	Нажата клавиша CapsLock
7	Нажата клавиша Insert

Если Вы изменяете состояние светодиодов на панели клавиатуры, не забывайте устанавливать соответствующие биты в байтах состояния клавиатуры.

Программой обработки прерывания INT 09h отслеживаются некоторые комбинации клавиш. В таблице приведены эти комбинации и действия, выполняемые обработчиком прерывания при их обнаружении:

Комбинация клавиш	*Выполняемые действия*
Ctrl-Alt-Del	Сброс и перезагрузка системы
Ctrl-NumLock, Pause	Перевод машины в состояние ожидания до нажатия любой клавиши
Shift-PrtSc	Распечатка на принтере содержимого видеопамяти
Ctrl-Break	Выполнение прерывания INT 1Bh, завершающего работу программы

Многие типы клавиатур имеют отдельную альтернативную цифровую панель, напоминающую клавиатуру калькулятора. Если одновременно с нажатием на клавишу Alt набрать число на этой панели (не большее чем 255 и не равное 0), то это число будет помещено в буфер клавиатуры, как будто бы оно было введено нажатием на одну клавишу. Это число будет также записано в слове по адресу 0000h:0419h в области данных BIOS.

При переполнении внутреннего буфера клавиатуры или буфера, расположенного в области данных BIOS программа-обработчик прерывания INT 09h генерирует звуковой сигнал.

При составлении программ для MS-DOS у Вас едва ли появится необходимость непосредственного манипулирования содержимым буфера клавиатуры - Вы можете использовать прерывание BIOS INT 16h для выполнения практически всех клавиатурных операций.

В следующем разделе мы займемся непосредственно изучением средств работы с клавиатурой, предоставляемых в распоряжение прерыванием BIOS INT 16h.

2.4. Средства BIOS для работы с клавиатурой

Набор функций для работы с клавиатурой, предоставляемый в распоряжение программиста прерыванием BIOS INT 16h, включает в себя функции для выборки кода нажатого символа из буфера с ожиданием нажатия, функции для проверки содержимого буфера и для управления содержимым буфера, функции для изменения скоростных характеристик клавиатуры.

2.4.1. Чтение символа с ожиданием

Функция 00h выполняет чтение кода символа из буфера клавиатуры, если он там есть. Если буфер клавиатуры пуст, программа переводится в состояние ожидания до тех пор, пока не будет нажата какая-нибудь клавиша. Скан-код и ASCII-код нажатой клавиши передаются программе.

Приведем формат вызова функции:

На входе: AH = 00h.

На выходе: AL = ASCII-код символа или 0, если AH содержит расширенный ASCII-код символа;

AH = скан-код или расширенный ASCII-код символа, если AL = 0.

Приведем таблицу скан-кодов для клавиатуры IBM PC/XT:

01	Esc	12	E	23	H	34	. >	45	NumLock
02	1 !	13	R	24	J	35	/ ?	46	ScrollLock
03	2 @	14	T	25	K	36	Shft(прав)	47	Home [7]
04	3 #	15	Y	26	L	37	* PrtSc	48	Up [8]
05	4 $	16	U	27	; :	38	Alt	49	PgUp [9]
06	5 %	17	I	28	" '	39	Пробел	4a	K -
07	6 ^	18	O	29	` ~	3a	CapsLock	4b	<- [4]
08	7 &	19	P	2a	Shft(лев)	3b	F1	4c	[5]
09	8 *	1a	[{	2b	\ |	3c	F2	4d	-> [6]
0a	9 (	1b	] }	2c	Z	3d	F3	4e	K +
0b	0)	1c	Enter	2d	X	3e	F4	4f	End [1]
0c	- _	1d	Ctrl	2e	C	3f	F5	50	Dn [2]
0d	+ =	1e	A	2f	V	40	F6	51	PgDn [3]
0e	Bksp	1f	S	30	B	41	F7	52	Ins [0]
0f	Tab	20	D	31	N	42	F8	53	Del [.]
10	Q	21	F	32	M	43	F9		
11	W	22	G	33	, <	44	F10		

Для остальных клавиш функция 00h прерывания INT 16h возвращает расширенный ASCII-код:

F1	3b	Shift-F1	54	Ctrl-F1	5e	Alt-F1	68
F2	3c	Shift-F2	55	Ctrl-F2	5f	Alt-F2	69
F3	3d	Shift-F3	56	Ctrl-F3	60	Alt-F3	6a
F4	3e	Shift-F4	57	Ctrl-F4	61	Alt-F4	6b
F5	3f	Shift-F5	58	Ctrl-F5	62	Alt-F5	6c
F6	40	Shift-F6	59	Ctrl-F6	63	Alt-F6	6d
F7	41	Shift-F7	5a	Ctrl-F7	64	Alt-F7	6e
F8	42	Shift-F8	5b	Ctrl-F8	65	Alt-F8	6f
F9	43	Shift-F9	5c	Ctrl-F9	66	Alt-F9	70
F10	44	Shift-F10	5d	Ctrl-F10	67	Alt-F10	71

Alt-A	1e	Alt-P	19	Alt-3	7a	Down Dn	50
Alt-B	30	Alt-Q	10	Alt-4	7b	Left <-	4b
Alt-C	2e	Alt-R	13	Alt-5	7c	Right ->	4d
Alt-D	20	Alt-S	1f	Alt-6	7d	Up Up	48
Alt-E	12	Alt-T	14	Alt-7	7e	End	4f
Alt-F	21	Alt-U	16	Alt-8	7f	Home	47
Alt-G	22	Alt-V	2f	Alt-9	80	PgDn	51
Alt-H	23	Alt-W	11	Alt--	82	PgUp	49
Alt-I	17	Alt-X	2d	Alt-=	83		
Alt-J	24	Alt-Y	15			^Left	73
Alt-K	25	Alt-Z	2c			^Right	74
Alt-L	26			Shift-Tab	0f	^End	75
Alt-M	32	Alt-0	81	Ins	52	^Home	77
Alt-N	31	Alt-1	78	Del	53	^PgDn	76
Alt-O	18	Alt-2	79	^PrtSc	72	^PgUp	84

В следующей таблице приведены ASCII-коды клавиш, имеющихся только на 101-клавишной клавиатуре:

F11	85	Alt-Bksp	0e	Alt- Д /	a4		
F12	86	Alt-Enter	1c	Alt- Д *	37		
Shft-F11	87	Alt-Esc	01	Alt- Д -	4a		
Shft-F12	88	Alt-Tab	a5	Alt- Д +	4e		
Ctrl-F11	89	Ctrl-Tab	94	Alt- Д Enter	a6		
Ctrl-F12	8a						
Alt-F11	8b	Alt-up	Up 98	Ctrl- Д /	95		
Alt-F12	8c	Alt-down	Dn a0	Ctrl- Д *	96		
Alt-[	1a	Alt-left	<- 9b	Ctrl- Д -	8e		
Alt-]	1b	Alt-right	-> 9d	Ctrl- Д +	90		
Alt-;	27						
Alt-'	28	Alt-Delete	a3	Ctrl- Д Up [8]	8d		
Alt-`	29	Alt-End	9f	Ctrl- Д 5 [5]	8f		
Alt-\	2b	Alt-Home	97	Ctrl- Д Dn [2]	91		
Alt-,	33	Alt-Insert	a2	Ctrl- Д Ins[0]	92		
Alt-.	34	Alt-PageUp	99	Ctrl- Д Del[.]	93		

Буква "Д" в последней таблице обозначает дополнительную ("калькуляторную") клавиатуру.

Для демонстрации использования функции 00h прерывания INT 16h мы подготовили программу, выводящую на экран сканкоды и ASCII-коды нажимаемых клавиш:

```
#include <stdio.h>
#include <dos.h>

void main(void);
```

```
void main(void) {

     union REGS  rg;

     printf("\nОпределение скан-кода и ASCII-кода клавиш."
       "\nДля завершения работы нажмите клавишу ESC.\n\n");

     for(;;) {

// Вызываем прерывание INT 16h

          rg.h.ah = 0;
          int86(0x16, &rg, &rg);

// Выводим на экран содержимое регистров AH и AL,
// содержащих соответственно скан-код и ASCII-код
// нажатой клавиши.

          printf("\nScan = %02.2X Ascii = %02.2X",
               rg.h.ah,
               rg.h.al);

// Если была нажата клавиша ESC, завершаем работу программы

          if(rg.h.ah == 1) break;

     }
}
```

2.4.2. Проверка буфера на наличие в нем символов

На входе:	AH	=	01h.
На выходе:	ZF	=	0, если в буфере имеется код нажатой на клавиатуре клавиши;
	ZF	=	1, если буфер клавиатуры пуст;
	AL	=	ASCII-код символа или 0, если AH содержит расширенный ASCII-код символа;
	AH	=	скан-код или расширенный ASCII-код символа, если AL=0.

Функция 01h поможет проверить состояние буфера клавиатуры - есть там коды нажатых клавиш или нет. При этом программа не переводится в состояние ожидания, даже если буфер клавиатуры пуст. В этом случае в регистре флагов устанавливается в единицу флаг ZF и управление возвращается программе.

Эту функцию удобно использовать во время выполнения какого-либо длительного процесса (например, форматирования диска или передачи данных по линии связи) для прерывания этого процесса по запросу оператора.

Кроме того, функцию можно использовать вместе с функцией 00h для сброса содержимого клавиатурного буфера. Для этого в цикле повторяют вызов функции 01h, вслед за которым идет вызов функции 00h при условии, что буфер клавиатуры не пуст. Сброс клавиатурного буфера полезно выполнять перед вводом ответственной информации, так как из-за случайного двойного или тройного нажатия на клавишу в буфере клавиатуры могут оказаться лишние символы.

Приведем текст программы, выводящей на экран в цикле символ '*'. При нажатии на любую клавишу, кроме ESC, программа выводит на экран строку текста - инструкцию для завершения работы программы. Если нажать на клавишу ESC, работа программы будет завершена.

```
#include <stdio.h>
#include <dos.h>

void main(void);

void main(void) {

     union REGS  rg;
     int   i, zflag;

     for(;;) {

// Выводим в цикле символ '*'

          putchar('*');

// Небольшая задержка во времени

          for(i=0; i<1000; i++);

// Вызываем прерывание INT 16h для проверки буфера клавиатуры

// Устанавливаем флаг, который будет сброшен при нажатии на
// любую клавишу

          zflag = 1;

          _asm {

               mov   ax, 0100h
               int   16h
```

```
// Если нажатия не было,
// продолжаем выполнение программы

                jz      nokey

// При нажатии на любую клавишу
// сбрасываем флаг

                mov     zflag, 0
nokey:

          }

          if(zflag == 0) {

// Если флаг сброшен, читаем код нажатой клавиши из буфера
// при помощи функции 01h прерывания INT 16h

                rg.h.ah = 0;
                int86(0x16, &rg, &rg);

// Если была нажата клавиша ESC, завершаем работу программы

                if(rg.h.ah == 1) {

// Выводим на экран содержимое регистров AH и AL,
// содержащих соответственно скан-код и ASCII-код
// нажатой клавиши.

                     printf("\nScan = %02.2X Ascii = %02.2X",
                          rg.h.ah,
                          rg.h.al);

                          break;
                }
                else printf("\nДля завершения нажмите ESC\n");
          }
     }
}
```

2.4.3. Получение состояния переключающих клавиш

На входе: AH = 02h.

На выходе: AL = байт состояния переключающих клавиш

Функция возвращает в регистре AL состояние переключающих клавиш (Shift, Ctrl, Alt, ScrollLock, NumLock, CapsLock, Ins). Формат байта состояния соответствует формату байта, находящегося в области данных BIOS по адресу 0000h:0417h:

Биты	Значение
0	Нажата правая клавиша Shift
1	Нажата левая клавиша Shift
2	Нажата комбинация клавиш Ctrl-Shift с любой стороны
3	Нажата комбинация клавиш Alt-Shift с любой стороны
4	Состояние клавиши ScrollLock
5	Состояние клавиши NumLock
6	Состояние клавиши CapsLock
7	Состояние клавиши Insert

Функция может быть использована для анализа текущего состояния переключающих клавиш.

Изменим текст предыдущей программы таким образом, чтобы завершение ее работы происходило лишь в том случае, если переключающая клавиша CapsLock находится в выключенном состоянии (соответствующий светодиод не горит):

```
#include <stdio.h>
#include <dos.h>

void main(void);

void main(void) {

     union REGS  rg;
     int   i, zflag;

     for(;;) {

// Выводим в цикле символ '*'

          putchar('*');

// Небольшая задержка во времени

          for(i=0; i<1000; i++);

// Вызываем прерывание INT 16h для проверки буфера клавиатуры

// Устанавливаем флаг, который будет сброшен при нажатии на
// любую клавишу

          zflag = 1;
```

```
        _asm {
             mov   ax, 0100h
             int   16h
// Если нажатия не было,
// продолжаем выполнение программы
             jz    nokey
// При нажатии на любую клавишу
// сбрасываем флаг
             mov   zflag, 0
nokey:
        }
        if(zflag == 0) {
// Если флаг сброшен, читаем код нажатой клавиши из буфера
// при помощи функции 01h прерывания INT 16h
             rg.h.ah = 0;
             int86(0x16, &rg, &rg);
// Если была нажата клавиша ESC, завершаем работу программы,
// при условии, что переключатель CapsLock выключен
             if(rg.h.ah == 1) {
// Дополнительно проверяем состояние клавиши CapsLock,
// этой клавише соответствует бит 0x40 в слове состояния
                  rg.h.ah = 2;
                  int86(0x16, &rg, &rg);
                  if((rg.h.al & 0x40) == 0) break;
                  else printf("\nДля завершения нажмите"
                    " ESC "
                    "при выключенной клавише CapsLock.\n");
             }
             else printf("\nДля завершения нажмите ESC "
                    "при выключенной клавише CapsLock.\n");
        }
    }
}
```

2.4.4. Установка временных характеристик клавиатуры

На входе: AH = 03h;
AL = 05h;

BL = период автоповтора (количество повторов за одну секунду):
0 - 30.0; 0Ah - 10.0;
1 - 26.7; 0Dh - 9.2;
2 - 24.0; 10h - 7.5;
4 - 20.0; 14h - 5.0;
8 - 15.0; 1Fh - 2.0;

BH = задержка включения режима автоповтора:
0 - 250 мс;
1 - 500 мс;
2 - 750 мс;
3 - 1000 мс.

На выходе: Не используются.

Мы уже рассказывали о возможности изменения временных характеристик клавиатуры. Если BIOS, установленная в Вашей машине, изготовлена после 15 декабря 1985 года, Вы можете воспользоваться этой функцией для ускорения (или замедления) работы клавиатуры.

В качестве примера приведем две программы. Первая увеличивает быстродействие клавиатуры до верхнего предела, вторая восстанавливает исходные значения временных характеристик.

```
#include <stdio.h>
#include <dos.h>

void main(void);

void main(void) {

     union REGS  rg;

          rg.h.al = 5;
          rg.h.ah = 3;

// Устанавливаем максимальное быстродействие клавиатуры

          rg.h.bl = 0;
          rg.h.bh = 0;

          int86(0x16, &rg, &rg);

}

#include <stdio.h>
#include <dos.h>
```

```
void main(void);

void main(void) {

     union REGS  rg;

          rg.h.al = 5;
          rg.h.ah = 3;

// Восстанавливаем исходное быстродействие клавиатуры

          rg.h.bl = 0xa;
          rg.h.bh = 1;

          int86(0x16, &rg, &rg);
}
```

2.4.5. Запись символов в буфер клавиатуры

На входе: AH = 05h;
CL = ASCII-код записываемого символа;
CH = скан-код записываемого символа или 0.

На выходе: AL = 0 - запись выполнена успешно;
1 - буфер клавиатуры переполнен.

С помощью этой функции можно вставить символы в буфер клавиатуры, как будто они были введены оператором.

Приведенная программа записывает в буфер клавиатуры пять символов '*'. Запустите ее и посмотрите на системное приглашение. Вы увидите что-нибудь похожее на C:\>*****.

```
#include <stdio.h>
#include <dos.h>

void main(void);

void main(void) {

     union REGS  rg;
     int   i;

      for(i=0; i<5; i++) {

          rg.h.ah = 5;

          rg.h.cl = '*';
          rg.h.ch = 9;

          int86(0x16, &rg, &rg);

      }
}
```

2.4.6. Чтение символа с ожиданием для 101-клавишной клавиатуры

Функция 10h полностью аналогична функции 00h, но она предназначена для работы с клавиатурой, имеющей 101 клавишу.

Приведем формат вызова функции:

На входе:	AH	=	10h.
На выходе:	AL	=	ASCII-код символа или 0, если AH содержит расширенный ASCII-код символа;
	AH	=	скан-код или расширенный ASCII-код символа, если AL = 0.

Функция определена для BIOS, изготовленной не раньше 15 декабря 1985 года.

2.4.7. Проверка буфера на наличие в нем символов для 101-клавишной клавиатуры

На входе:	AH	=	11h.
На выходе:	ZF	=	0, если в буфере имеется код нажатой на клавиатуре клавиши;
	ZF	=	1, если буфер клавиатуры пуст;
	AL	=	ASCII-код символа или 0, если AH содержит расширенный ASCII-код символа;
	AH	=	скан-код или расширенный ASCII-код символа, если AL = 0.

Функция 11h полностью аналогична функции 01h, но она предназначена для работы с клавиатурой, имеющей 101 клавишу.

Эта функция определена для BIOS, изготовленной не раньше 15 декабря 1985 года.

2.4.8. Получение состояния переключающих клавиш для 101-клавишной клавиатуры

На входе:	AH	=	12h.
На выходе:	AL	=	байт состояния переключающих клавиш.

Функция возвращает в регистре AL состояние переключающих клавиш (Shift, Ctrl, Alt, ScrollLock, NumLock, CapsLock, Ins):

Биты	*Значение*
0	Нажата левая клавиша Shift вместе с Ctrl
1	Нажата левая клавиша Shift вместе с Alt
2	Нажата правая клавиша Shift вместе с Ctrl
3	Нажата правая клавиша Shift вместе с Alt
4	Нажата клавиша ScrollLock
5	Нажата клавиша NumLock
6	Нажата клавиша CapsLock
7	Нажата клавиша SysReq

Функция 12h аналогична функции 02h, но она предназначена для работы с клавиатурой, имеющей 101 клавишу, и имеет другой формат байта состояния. Эта функция определена для BIOS, изготовленной не раньше 15 декабря 1985 года.

2.5. Средства MS-DOS для работы с клавиатурой

К сожалению, MS-DOS не предоставляет программам каких-либо существенных дополнительных возможностей по сравнению с функциями прерывания BIOS INT 16h. Поэтому многие программы работают с клавиатурой через BIOS.

Однако если Ваша программа пользуется клавиатурными функциями MS-DOS, то ей доступно средство переназначения ввода операционной системы. Это возможно благодаря тому, что клавиатурные функции MS-DOS являются функциями, работающими со стандартным вводом MS-DOS, а стандартный ввод может быть переназначен.

Кроме того, некоторые клавиатурные функции автоматически посылают введенные символы на устройство стандартного вывода. По умолчанию это дисплей, но устройство стандартного вывода может быть переназначено для вывода в файл, на принтер или другое устройство.

Вообще говоря, клавиатурные функции MS-DOS больше всего подходят для тех программ, которые ведут с оператором "построчный" диалог. Для таких программ при использовании средств переназначения ввода/вывода возможна организация автоматического "пакетного" выполнения, когда все сообщения выводятся в файл, а все данные, которые обычно вводятся с клавиатуры, считываются из заранее подготовленного файла "ответов".

Некоторые клавиатурные функции MS-DOS отслеживают комбинации клавиш Ctrl-C и Ctrl-Break. Если оператор ввел такую комбинацию клавиш, вызывается прерывание INT 23h, завершающее работу текущей программы. Если Ваша программа не должна завершаться при нажатии этих комбинаций клавиш, можно либо создать и подключить собственный обработчик для INT 23h, либо использовать те клавиатурные функции MS-DOS, которые не выполняют проверку указанных выше комбинаций клавиш.

Приведем подробное описание клавиатурных функций прерывания MS-DOS INT 21h.

2.5.1. Буферизованный ввод с эхо-выводом

На входе:	AH	=	01h.
На выходе:	AL	=	ASCII-код символа или 0. Если регистр содержит 0, то следующий вызов этой же функции возвратит в регистре AL расширенный ASCII-код символа. Функция проверяет комбинации клавиш Ctrl-C и Ctrl-Break.

Функция читает символы со стандартного устройства ввода. Если стандартным устройством ввода является клавиатура и буфер клавиатуры пуст, выполнение программы задерживается до нажатия на любую клавишу.

Введенный символ выводится на стандартное устройство вывода.

Если программа в качестве ASCII-кода получила 0, она должна вызвать эту функцию еще один раз. Во второй раз регистр AL будет содержать расширенный ASCII-код нажатой клавиши.

2.5.2. Буферизованный ввод без эхо-вывода

На входе:	AH	=	08h.
На выходе:	AL	=	ASCII-код символа или 0. Если регистр содержит 0, то следующий вызов этой же функции возвратит в регистре AL расширенный ASCII-код символа. Функция проверяет комбинации клавиш Ctrl-C и Ctrl-Break.

Функция аналогична предыдущей. Она читает символы со стандартного устройства ввода. Если стандартным устройством ввода является клавиатура и буфер клавиатуры пуст, выполнение программы задерживается до нажатия на любую клавишу.

Эту функцию необходимо использовать в тех случаях, когда не требуется автоматически дублировать на экране вводимые с клавиатуры символы. Например, с ее помощью можно организовать ввод паролей.

2.5.3. Нефильтрованный ввод без эхо-вывода

На входе:	AH	=	07h.
На выходе:	AL	=	ASCII-код символа или 0. Если регистр содержит 0, то следующий вызов этой же функции возвратит в регистре AL расширенный ASCII-код символа. Функция не проверяет комбинации клавиш Ctrl-C и Ctrl-Break.

Если буфер клавиатуры пуст, выполнение программы задерживается до нажатия на любую клавишу.

Эту функцию удобно использовать в тех случаях, когда завершение программы по нажатию комбинаций клавиш Ctrl-C или Ctrl-Break по тем или иным причинам нежелательно. Например, программа держит в оперативной памяти буфера для данных, которые перед завершением работы обязательно должны быть записаны на диск. Если оператор в неподходящий момент нажал Ctrl-C и программа аварийно завершила работу, содержимое буферов будет потеряно.

2.5.4. Ввод/вывод на консоль

На входе:	AH	=	06h;
	DL	=	0FFh - для ввода символа с консоли; или:
	DL	=	код символа, не равный 0FFh, - для вывода символа на консоль.
На выходе:	ZF	=	0, если в буфере имеется код нажатой на клавиатуре клавиши;
	ZF	=	1, если буфер клавиатуры пуст;
	AL	=	ASCII-код символа или 0, если AH содержит расширенный ASCII-код символа. Функция проверяет комбинации клавиш Ctrl-C и Ctrl-Break.

Функция 06h может использоваться как для ввода с консоли, так и для вывода символов на консоль. Режим работы функции зависит от содержимого регистра DL при вызове функции. Если этот регистр содержит значение 0FFh, функция выполняет ввод с консоли, в противном случае символ, код которого записан в этот регистр, выводится на консоль.

Очевидно, что с помощью этой функции нельзя вывести на консоль символ с кодом 0FFh.

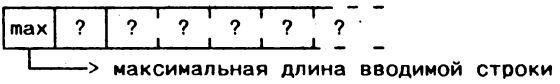
Основное отличие функции 06h от всех описанных ранее заключается в том, что эта функция не ожидает, пока оператор нажмет на клавишу. Если буфер клавиатуры пуст, функция просто устанавливает флаг процессора ZF в 1.

Если в буфере клавиатуры имеются символы, флаг ZF сбрасывается и в регистр AL функция записывает ASCII-код символа.

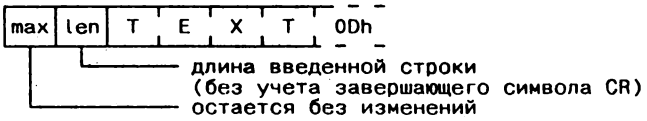
2.5.5. Ввод строки символов

На входе:	AH	=	0Ah;
	DS:DX	=	адрес буфера для ввода строки.
На выходе:	Буфер содержит введенную строку. Функция проверяет комбинации клавиш Ctrl-C и Ctrl-Break.		

Функция предназначена для ввода с клавиатуры строки символов. Перед вызовом функции необходимо специальным образом подготовить буфер, адрес которого передается в регистрах DS:DX, - в первый байт буфера следует записать максимальную длину вводимой строки (в диапазоне от 1 до 244):



После возврата из функции буфер будет иметь следующий формат:



Ввод осуществляется до тех пор, пока либо количество введенных символов не достигнет max-1, либо пока не будет нажата клавиша Enter (код 0Dh). Если оператор уже ввел max-1 символ и продолжает вводить символы дальше, функция выдает звуковой сигнал на каждое нажатие и игнорирует вводимые символы до тех пор, пока не будет нажата клавиша Enter.

При вводе строки можно использовать стандартные средства редактирования MS-DOS, используемые при вводе команд в режиме командной строки.

2.5.6. Проверка состояния стандартного ввода

На входе: AH = 0Bh.

На выходе: AL = 0FFh, если в буфере имеется код нажатой на клавиатуре клавиши;
AL = 0, если буфер клавиатуры пуст.
Функция проверяет комбинации клавиш Ctrl-C и Ctrl-Break.

Эта функция проверяет состояние клавиатурного буфера. Вы можете вызывать ее перед функциями 01h, 07h, 08h для того, чтобы избежать ожидания нажатия на клавишу.

Если программа выполняет какую-либо длительную обработку (копирование файлов, форматирование дисков и т. п.), Вы можете вызывать эту функцию в процессе обработки для проверки нажатия комбинации клавиш, прерывающих работу программы.

2.5.7. Сброс буфера клавиатуры

На входе: AH = 0Ch;
AL = 1, 6, 7, 8 или 0Ah.

На выходе: Не определены.

Функция очищает клавиатурный буфер и вызывает клавиатурную функцию MS-DOS, номер которой определяется содержимым регистра AL. Если же регистр AL содержит другое значение, кроме приведенных выше, функция просто сбрасывает содержимое буфера и не выполняет никаких других действий.

Эту функцию удобно использовать тогда, когда перед вводом символа необходимо убедиться в том, что буфер клавиатуры пуст.

2.6. Клавиатурные функции библиотеки Microsoft C

Стандартные библиотеки трансляторов Microsoft QuickC и C 6.0 содержат набор функций, предназначенных для работы с клавиатурой. Эти функции повторяют и немного дополняют возможности функций MS-DOS и BIOS, обслуживающих клавиатуру.

Самые простые из них - getch() и getche(). Они описаны в файле conio.h.

Функция getch() имеет следующий прототип:

```
int getch(void);
```

Эта функция возвращает ASCII-код прочитанного из клавиатурного буфера символа, причем прочитанный символ не отображается на экране. Если была нажата функциональная клавиша или клавиша перемещения курсора, функция возвращает 0. В этом случае функцию надо вызвать еще раз для получения расширенного ASCII-кода нажатой клавиши.

Функция обрабатывает клавиши Ctrl-C и Ctrl-Break - при вводе этих комбинаций клавиш работа программы завершается.

Если клавиатурный буфер пуст, программа переводится в состояние ожидания.

Функция getche() полностью аналогична функции getch(), за исключением того, что прочитанный символ отображается на экране. Приведем прототип функции getche():

```
int getche(void);
```

Приведем пример программы, отображающей на экране ASCII-коды и расширенные ASCII-коды нажимаемых клавиш:

```
#include <conio.h>
#include <ctype.h>
#include <stdio.h>

void main() {
      int key;

// Читаем в цикле символы с клавиатуры и отображаем
// ASCII-коды нажатых клавиш.
// Выходим из цикла при нажатии не клавишу ESC

      for(;;) {

// Читаем символ

            key = getch();

// Если прочитанный символ равен 0, вызываем функцию getch()
// для получения расширенного ASCII-кода нажатой клавиши

            if( (key == 0) || (key == 0xe0) ) {
                  key = getch();
                  printf( "Расширенный ASCII-код:\t" );
            }

            else printf( "ASCII-код:\t");

            printf("%d\n",key);

// При нажатии на клавишу ESC выходим из цикла

            if( key == 27) break;
      }
}
```

Для проверки буфера клавиатуры на наличие символов можно использовать функцию kbhit(). Она описана в файле conio.h:

```
int kbhit(void);
```

Если буфер клавиатуры не пуст, функция возвращает ненулевое значение. В этом случае программа может прочитать символы из буфера клавиатуры при помощи функций getch() и getche(). Если буфер пуст, функция возвращает нулевое значение.

Приведем пример программы, ожидающей нажатия на любую клавишу. Во время ожидания программа выводит на экран поочередно символы '<' и '>':

```
#include <conio.h>

void main() {

     int key;

// Ожидаем нажатия на любую клавишу.
// Во время ожидания выводим на экран поочередно
// символы '<' и '>'

     while(!kbhit()) printf("<\b>\b");

// Как только будет нажата какая-нибудь клавиша,
// выводим ее ASCII-код

          key = getch();

// Если прочитанный символ равен 0, вызываем функцию getch()
// для получения расширенного ASCII-кода нажатой клавиши

          if( (key == 0) || (key == 0xe0) ) {
                key = getch();
                printf( "Расширенный ASCII-код:\t" );
          }

          else printf( "ASCII-код:\t");

          printf("%d\n",key);
}
```

Для ввода с клавиатуры строки символов можно использовать функцию cgets(), работающую аналогично функции 0Ah прерывания MS-DOS INT 21h:

```
char *cgets(char *buffer);
```

Функция описана в файле conio.h. Перед вызовом аргумент функции buffer должен указывать на массив, размер которого должен быть достаточным для хранения вводимой строки, завершающего строку нулевого байта и двух дополнительных байтов. Первый элемент массива buffer[0] должен содержать максимальную длину вводимой строки - как и для функции 0Ah прерывания MS-DOS INT 21h.

После завершения ввода второй элемент массива buffer[1] будет содержать длину введенной строки, сама строка будет завершаться символами новой строки NL, перевода строки LF и нулем.

Функция cgets() возвращает указатель на начало введенной строки в буфере, т. е. на третий элемент массива buffer[2].

Приведем простой пример, в котором функция cgets() используется для ввода целого числа:

```
#include <stdlib.h>
#include <string.h>
#include <conio.h>

#define MAX 80

char buf[MAX];

void main() {
     int     i;
     char    *bufptr;

// Устанавливаем максимально допустимую длину строки

     buf[0] = MAX +.2;

     printf("\nВведите целое число: ");

// Вводим число, можно использовать клавиши редактирования

     bufptr = cgets(buf);

// Преобразуем введенное число к формату int
// и выводим его

     i = atoi(bufptr);
     printf("\nВы ввели число %d", i);
}
```

Существует и более удобная для использования функция, позволяющая вводить строку с клавиатуры, а точнее, из стандартного потока ввода. Это функция gets():

```
char *gets(char *buffer);
```

Функция gets() описана в файле stdio.h.

Эта функция читает строку из стандартного потока ввода stdin и запоминает ее в буфере buffer. Символ новой строки '\n' в конце введенной строки функция заменяет на ноль.

После завершения ввода функция возвращает указатель на заполненный буфер или NULL в случае ошибки или условия "Конец файла".

Обратите внимание на отличия между функциями cgets() и gets():

- Функция cgets() позволяет редактировать вводимую строку символов, функция gets() просто записывает в буфер все символы подряд (в том числе и коды клавиш редактирования).
- Программе, использующей для ввода с клавиатуры функцию cgets(), недоступны средства переназначения ввода операционной системы. Если же программа использует функцию gets(), читающую строку из стандартного потока ввода, можно использовать средства переназначения.
- Перед вызовом функции cgets() необходимо специальным образом подготовить буфер для вводимой строки (записать в первый байт буфера длину вводимой строки). Функция gets() не требует никакой подготовки буфера.

Еще одна полезная функция, которую можно использовать для ввода с клавиатуры, - scanf(). Эта функция подробно описана во всех книгах по языку программирования Си, поэтому мы не будем ее подробно рассматривать. Отметим только, что с помощью этой функции можно организовать ввод чисел в заданном формате. Однако можно сначала ввести строку при помощи функций cgets() или gets(), а уже потом выполнять все необходимые проверки и преобразования этой строки.

Глава 3

МЫШЬ

Вместе с появлением персональных компьютеров возникло и получило огромную популярность графическое устройство ввода информации - мышь. В настоящее время практически каждый персональный компьютер оснащен этим устройством. Более того, многие программы специально ориентированы на использование мыши. Например, графический редактор Picture Maker из пакета Story Editor. В этом редакторе Вы можете сделать почти всю работу, не прикасаясь к клавиатуре (за исключением операции набора текстовых строк). Что это за устройство и почему оно используется так же часто, как и клавиатура персонального компьютера?

3.1. Как устроена мышь?

Мышь - это небольшая коробочка с двумя или тремя клавишами, которая соединяется с компьютером тонким кабелем:

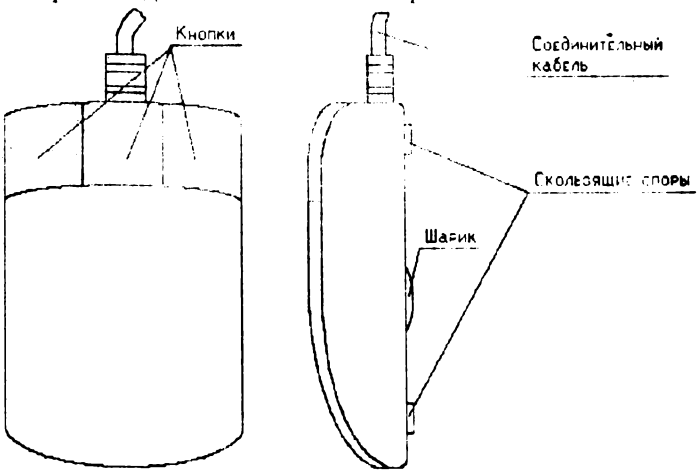


Рис. 2. Внешний вид мыши

Сверху на корпусе расположены кнопки. Обычно их две или три. Назначение этих кнопок полностью определяется программным обеспечением. Снизу виден шарик. Он обычно покрыт резиной для лучшего сцепления с поверхностью стола.

Все, что Вам нужно делать с мышью, - это катать ее по любой гладкой поверхности и нажимать на кнопки. Программное обеспечение свяжет перемещения мыши по поверхности стола с перемещениями, например, курсора по поверхности экрана. Перемещая мышь по столу (и соответственно курсор по экрану), Вы можете указывать (выбирать) различные объекты, находящиеся на экране.

Для того, чтобы "выбрать" какой-нибудь объект, обычно требуется указать на этот объект курсором и нажать на одну из кнопок мыши.

Если Вы откросте корпус мыши, Вы увидите простой механизм, состоящий из шарика, двух осей с резиновыми валиками, двух дисков с отверстиями и четырех фотодатчиков:

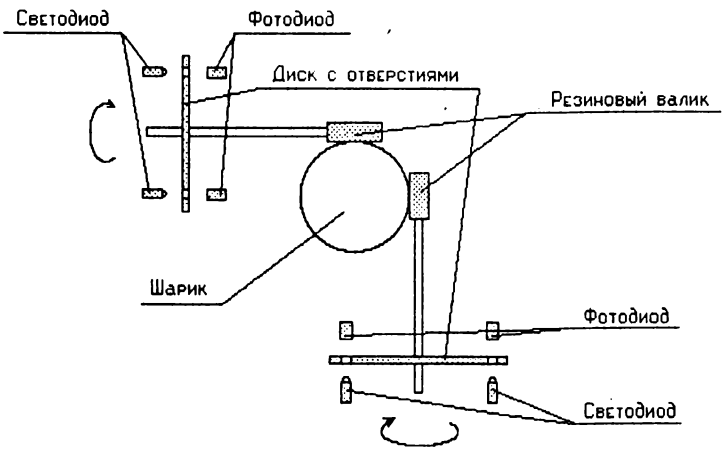


Рис. 3. Внутреннее устройство мыши

Когда Вы перемещаете мышь по поверхности стола, вращение шарика передается через резиновые валики двум дискам с отверстиями. Около каждого диска расположены фотодатчики (по два на диск). Они фиксируют направление вращения и угол поворота дисков. Во время движения мыши фотодатчики вырабатывают импульсы, которые передаются в компьютер. Количество этих импульсов линейно зависит от величины перемещения мыши.

В настоящее время изготовители компьютерного оборудования предлагают мыши разного типа, которые отличаются не только внешним видом и количеством клавишей, но также и способом подключения к компьютеру, могут иметь различную точность и различный программный интерфейс.

Можно выделить два наиболее часто используемых способа подключения мыши к компьютеру:

- через последовательный порт (COM1, COM2);
- через специальный адаптер, который вставляется в слот расширения материнской платы компьютера.

Выбирая мышь, используйте тот способ подключения, который Вам более удобен. Существуют компьютеры, не имеющие слотов расширения (обычно это компьютеры типа Lap-Top). Для таких компьютеров больше подойдет мышь, подключающаяся через последовательный порт.

Что касается программного интерфейса, то можно выделить два типа:

- трехкнопочная мышь системы Mouse Systems;
- двухкнопочная мышь Microsoft.

Некоторые мыши могут эмулировать оба типа. Эмулируемый тип зависит от состояния переключателя, находящегося на нижней крышке корпуса мыши или от того, была ли нажата клавиша мыши во время включения питания компьютера.

Мы рекомендуем Вам приобрести двухкнопочную мышь фирмы Microsoft. Эта мышь удобна в работе, имеет переходник для подключения к последовательному порту и адаптер для подключения к слотам расширения. Кроме того, если Вы используете мышь фирмы Microsoft, есть гарантия, что все фирменное программное обеспечение будет правильно работать с Вашей мышью.

3.2. Драйверы мыши в MS-DOS

Как это ни странно, ни BIOS, ни MS-DOS версий вплоть до 4.01 не содержат программной поддержки мыши. Для того чтобы задействовать это устройство, Вам надо использовать драйвер мыши или специальную резидентную программу, выполняющую функцию драйвера мыши. Как правило, это программное обеспечение поставляется вместе с мышью.

Для подключения драйвера мыши файл CONFIG.SYS должен содержать строку следующего вида:

```
device=c:\mouse\mouse.sys
```

Если используется резидентная программа, она обычно вызывается в файле AUTOEXEC.BAT:

```
c:\mouse\mouse.com
```

Драйвер мыши выполняет следующие функции:

- отслеживает перемещения курсора и нажатия на клавиши мыши;
- рисует на экране курсор, повторяющий движения мыши в графическом или текстовом режимах;
- предоставляет программам интерфейс для работы с мышью, основанный на вызове прерывания INT 33h.

3.3. Прерывание для обслуживания мыши

Драйвер мыши, независимо от того, реализован он через устанавливаемый драйвер или резидентную программу, определяет обработчик прерывания INT 33h. Этот обработчик выполняет все операции, связанные с обслуживанием мыши:

- сброс мыши и установка драйвера в исходное состояние;
- включение/выключение курсора мыши;
- установка курсора в определенное место экрана;
- определение текущих координат курсора и текущего состояния клавиш;
- определение координат курсора и состояния клавиш в момент нажатия и в момент отпускания клавиши;
- определение области на экране, в пределах которой может перемещаться курсор;

- определение области на экране, в пределах которой курсор не будет виден;
- определение формы графического и текстового курсоров;
- определение величины перемещения мыши в сотых долях дюйма;
- подключение к драйверу пользовательской процедуры, получающей управление при нажатии на заданную клавишу или при перемещении мыши;
- запоминание и восстановление состояния драйвера;
- управление эмуляцией светового пера;
- управление скоростью движения курсора;
- задание/определение используемой видеостраницы;
- управление драйвером мыши.

Приведем подробное описание всех функций прерывания INT 33h, используемых при работе с мышью.

3.3.1. Инициализация мыши

На входе:	AX	=	0000h.
На выходе:	AX	=	состояние мыши: 0000h - драйвер мыши или мышь не установлены; FFFFh - драйвер и мышь установлены;
	BX	=	количество клавиш у мыши: 2 - две клавиши; 0 - больше или меньше чем две; 3 - мышь системы Mouse Systems (имеет три клавиши).

Эта функция выполняет аппаратный сброс оборудования мыши и программную установку драйвера мыши в начальное состояние. С помощью функции 21h можно установить драйвер в исходное состояние, не выполняя аппаратного сброса мыши.

При установке в исходное состояние для программ, работающих в текстовом режиме, выполняются следующие действия:

- курсор перемещается в центр экрана и гасится;

- разрешается перемещение курсора по всей поверхности экрана, причем на экране отсутствуют зоны, в которых курсор является невидимым;
- устанавливается режим отображения курсора - инвертирование атрибута символа, на который указывает курсор;
- для изображения курсора выбирается нулевая страница видеопамяти;
- разрешается эмуляция светового пера (хотя это Вам едва ли понадобится);
- задается начальная скорость перемещения курсора.

Мы подготовили функцию для инициализации мыши из программы, составленной на языке Си:

```
/**
*.Name      ms_init
*.Title     Инициализация мыши
*
*.Descr     Эта функция выполняет аппаратный сброс мыши,
*           устанавливает в начальные значения внутренние
*           переменные ее драйвера. Дополнительно определяется
*           количество клавиш мыши.
*
*.Proto     int ms_init(int *nbottoms)
*
*.Params    int   *nbottoms - указатель на переменную
*               типа int, в которую будет записано количество
*               клавиш, имеющихся в мыши.
*
*.Return    0   - плата или драйвер не установлены;
*           -1  - плата установлена, инициализация
*                 выполнена успешно;
*
*           В переменную nbottoms записывается количество
*           клавиш мыши:
*
*            2  - две клавиши;
*            0  - больше или меньше чем две;
*            3  - мышь системы Mouse Systems, три клавиши.
*
*.Sample    ms_sampl1.c
**/

#include <dos.h>
#include <conio.h>

union REGS reg;
```

```
int ms_init(int *nbottoms) {

     reg.x.ax = 0;
     int86(0x33,&reg,&reg);

     *nbottoms = reg.x.bx;
     return reg.x.ax;
}
```

3.3.2. Включить курсор мыши

На входе: AX = 0001h.

На выходе: Регистры не используются.

Для управления видимостью курсора драйвер мыши использует внутренний счетчик. Этот счетчик можно увеличивать, вызывая функцию 01h прерывания INT 33h, или уменьшать при помощи функции 02h этого же прерывания.

После инициализации драйвера функцией 00h счетчик устанавливается равным -1. После первого вызова функции 01h счетчик становится равным нулю. При этом курсор мыши становится видимым, его можно перемещать по экрану.

Если счетчик равен нулю, то следующие вызовы функции 01h игнорируются драйвером. Для того чтобы погасить курсор, используйте функцию 02h, которая при вызове уменьшает каждый раз содержимое счетчика на единицу.

Функция 01h сбрасывает область, в которой курсор не отображается (если такая область была ранее установлена функцией 10h).

Вызов функции из Си:

```
/**
*.Name       ms_on
*.Title      Включение курсора мыши
*
*.Descr      Функция увеличивает на 1 индикатор уровня видимости
*            курсора. Если индикатор равен нулю, курсор появля-
*            ется на экране. Значение индикатора не превышает
*            нуля даже при многократных вызовах этой функции.
*
*.Proto      void ms_on(void)
*
*.Params     Не используются
*
```

```
*.Return    Ничего
*
*.Sample    ms_sampl1.c
**/

#include <dos.h>
#include <conio.h>

union REGS reg;

void ms_on(void) {

     reg.x.ax = 1;
     int86(0x33,&reg,&reg);
}
```

3.3.3. Выключить курсор мыши

На входе: AX = 0002h.

На выходе: Регистры не используются.

Эта функция уменьшает на единицу счетчик видимости курсора. Если содержимое счетчика становится равным -1, изображение курсора пропадает с экрана.

Если Ваша программа использует для вывода на экран метод прямой записи в экранную память, перед обновлением содержимого экрана необходимо погасить курсор, а после завершения обновления высветить его опять. Это связано с тем, что драйвер мыши "помнит" старое значение атрибута символа, на который указывал курсор до обновления содержимого видеопамяти. Вы изменили атрибут, записав новое значение непосредственно в экранную память. Теперь, если установить курсор мыши на другой символ, изображение старого символа будет испорчено - появится прямоугольник (как бы еще одно изображение курсора мыши).

Вызов функции:

```
/**
*.Name      ms_off
*.Title     Выключение курсора мыши
*
*.Descr     Эта функция уменьшает на 1 индикатор уровня
*           видимости курсора. После вызова этой функции
*           курсор, если он был на экране, исчезает.
*           Многократные обращения будут последовательно
*           уменьшать индикатор и затем потребуют многократных
*           вызовов функции ms_on для его включения.
*
```

```
*.Proto      void ms_off(void)
*
*.Params     Не используются
*
*.Return     Ничего
*
*.Sample     ms_sampl1.c
**/

#include <dos.h>
#include <conio.h>

union REGS reg;

void ms_off(void) {

     reg.x.ax = 2;
     int86(0x33,&reg,&reg);
}
```

Приведем программу, в которой применяются рассмотренные выше функции. Она инициализирует мышь, делает видимым курсор мыши и "прячет" курсор после нажатия на любую клавишу:

```
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

void main() {

     int botm;

// Инициализируем мышь, определяем количество клавиш

     if(!ms_init(&botm)) {
          printf("\nМышь не установлена");
          exit(-1);
     }

     printf("\nУстановлена мышь: ");
     switch (botm) {
       case 2:
               printf("двухклавишная"); break;
       case 3:
               printf("трехклавишная, системы Mouse Systems");
               break;
       case 0:
       default:
               printf("неизвестной системы"); break;
     }

// Включаем курсор и ожидаем нажатия на клавишу

     printf("\n\nКурсор мыши включен, "
          "для выключения нажмите любую клавишу");
```

```
    ms_on();
    getch();
// Выключаем курсор
    ms_off();
    printf("\nКурсор выключен, "
         "для завершения нажмите любую клавишу");
    getch();
}
```

3.3.4. Определить положение курсора

На входе: AX = 0003h.

На выходе: BX = состояние клавиш мыши:
бит 0 = 1 - нажата левая клавиша;
бит 1 = 1 - нажата правая клавиша;
бит 2 = 1 - нажата средняя клавиша
(для мыши системы Mouse Systems);
CX = координата X (по горизонтали);
DX = координата Y (по вертикали).

Функция 03h возвращает текущие (на момент вызова функции) координаты курсора мыши и состояние клавиш.

Для графических режимов координаты располагаются в различных диапазонах, в зависимости от текущего видеорежима:

Размер экрана	*Номер режима*	*Диапазон координат* X	Y
320x200	4,5	0..638	0..199
640x200	6	0..639	0..199
320x200	0Dh	0..638	0..199
640x200	0Eh	0..639	0..199
640x350	0Fh	0..639	0..349

Программы, работающие в текстовом режиме, должны разделить полученные координаты на 8 (как координату X, так и координату Y).

Приведем функцию, предназначенную для использования в программах, составленных на языке Си:

```
/**
*.Name       ms_query
*.Title      Определение текущих координат курсора
*
*.Descr      Эта функция определяет текущие координаты курсора
*            мыши и состояние клавиш на момент вызова.
*            Определенное состояние записывается в структуру
*            MOUSE_STATE, описанную в файле sysp.h:
*
*              typedef struct _MOUSE_STATE_ {
*                    unsigned bottoms;
*                       unsigned x;
*                       unsigned y;
*                  } MOUSE_STATE;
*
*            Адрес структуры передается функции в качестве
*            параметра.
*
*.Proto      MOUSE_STATE *ms_query(MOUSE_STATE *state);
*
*.Params     MOUSE_STATE *state - указатель на структуру,
*                          описывающую состояние мыши.
*
*.Return     Функция возвращает значение своего параметра.
*
*.Sample     ms_samp1.c
**/

#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include "sysp.h"

union REGS reg;

MOUSE_STATE *ms_query(MOUSE_STATE *state) {

     reg.x.ax = 3;
     int86(0x33,&reg,&reg);

     state->bottoms = reg.x.bx;
     state->x       = reg.x.cx;
     state->y       = reg.x.dx;

     return(state);
}
```

Приведем пример программы, которая запрашивает номер видеорежима, устанавливает его и динамически отображает координаты курсора и состояние клавиш мыши. После завершения работы программа восстанавливает первоначальный видеорежим:

"ДИАЛОГ-МИФИ"

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

union REGS reg;
void main() {

     int botm, i;
     MOUSE_STATE state;
     unsigned old_videomode, new_videomode;
     char buf[20], *bufptr;


// Определяем текущий видеорежим

     reg.x.ax = 0x0f00;
     int86(0x10, &reg, &reg);
     old_videomode = reg.h.al;

// Устанавливаем новый видеорежим:

// Устанавливаем максимально допустимую длину строки

     buf[0] = 10;
     printf("\nВведите десятичный номер видеорежима: ");
     bufptr = cgets(buf);

// Преобразуем введенное число к формату int

     new_videomode = atoi(bufptr);

     reg.h.ah = 0;
     reg.h.al = new_videomode;
     int86(0x10, &reg, &reg);

// Инициализируем мышь, определяем количество клавиш

     if(!ms_init(&botm)) {
          printf("\nМышь не установлена");
          exit(-1);
     }

     printf("\nУстановлена мышь: ");
     switch (botm) {
       case 2:
               printf("двухклавишная"); break;
       case 3:
               printf("трехклавишная, системы Mouse Systems");
               break;
       case 0:
       default:
               printf("неизвестной системы"); break;
     }
     printf("\n\nСостояние мыши:\n\n");
```

```
// Включаем курсор
     ms_on();
     while(!kbhit()) {
          ms_query(&state);
          printf("%2d x:%5d y:%5d",
               state.bottoms,
               state.x,
               state.y);
          for(i=0;i<18;i++) printf("\b");
     }
     getch();
     ms_off();
     reg.h.ah = 0;
     reg.h.al = old_videomode;
     int86(0x10, &reg, &reg);
}
```

Однако использование функции 03h не самый лучший способ работы с мышью. Программа должна постоянно следить за координатами курсора или за состоянием клавиш. Это может привести к непроизводительным затратам процессорного времени на опрос состояния.

Немного позже мы рассмотрим другие способы определения состояния мыши.

3.3.5. Установить курсор

На входе: AX = 0004h;
CX = устанавливаемая координата X (по горизонтали);
DX = устанавливаемая координата Y (по вертикали).

На выходе: Регистры не используются.

Обычно курсор мыши устанавливает оператор. Но с помощью функции 04h программа тоже может установить курсор в заданную позицию. Для текстового режима устанавливаемые номера строки и столбца должны быть умножены на 8.

Если программа пытается установить курсор в область, где курсор невидим (эта область задается функцией 10h), то она смо-

жет это сделать. Курсор при этом исчезнет с экрана, что не всегда желательно.

Если при помощи функций 07h или 08h область для перемещения курсора была ограничена, то при попытке установить курсор за границу этой области, он будет установлен в точку, которая находится внутри границы и находится на минимальном расстоянии от точки, заданной при вызове функции.

Функция для установки курсора:

```
/**
*.Name      ms_setcr
*.Title     Установка курсора в заданную точку
*
*.Descr     Эта функция выполняет установку курсора мыши
*           в точку, заданную координатами X и Y.
*
*.Proto     void ms_setcr(int x, int y)
*
*.Params    int x   - горизонтальная координата курсора;
*           int y   - вертикальная координата курсора.
*
*.Return    Ничего
*
*.Sample    ms_samp2.c
**/

#include <dos.h>
#include <conio.h>

union REGS reg;

void ms_setcr(int x, int y) {

     reg.x.ax = 4;
     reg.x.cx = x;
     reg.x.dx = y;

     int86(0x33,&reg,&reg);
}
```

Приведем пример простой программы, которая устанавливает курсор в левый верхний угол экрана:

```
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

void main() {

     int botm;
```

```
// Инициализируем мышь
     if(!ms_init(&botm)) {
          printf("\nМышь не установлена");
          exit(-1);
     }
// Включаем курсор и ожидаем нажатия на клавишу
     printf("\n\nКурсор мыши включен, "
               "для выключения нажмите любую клавишу");
     ms_on();
// Устанавливаем курсор в левый верхний угол экрана
     ms_setcr(0,0);
     getch();
// Выключаем курсор
     ms_off();
}
```

3.3.6. Определить положение курсора при нажатии клавиши

На входе:	AX	=	0005h;
	BX	=	клавиша, при нажатии которой запоминается состояние мыши: 0 - левая; 1 - правая; 2 - средняя.
На выходе:	AX	=	состояние клавиш мыши: бит 0 = 1 - нажата левая клавиша; бит 1 = 1 - нажата правая клавиша; бит 2 = 1 - нажата средняя клавиша (для мыши системы Mouse Systems);
	BX	=	количество нажатий на заданную клавишу, после вызова функции в регистр записывается нуль;
	CX	=	координата X (по горизонтали);
	DX	=	координата Y (по вертикали).

В отличие от функции 03h эта функция возвращает программе не текущее состояние мыши, а запомненное в момент последнего нажатия на клавишу, заранее определенную при вызове функции. Она также возвращает количество нажатий на заданную клавишу, которое Вы можете использовать для обнаружения двойных нажатий.

Функция для определения состояния мыши при нажатии на заданную клавишу:

```
/**
*.Name       ms_querp
*.Title      Определение состояния мыши при нажатии на клавишу
*
*.Descr      Эта функция определяет координаты курсора мыши
*            и состояние клавиш в момент нажатия на заранее
*            заданную клавишу.
*            Определенное состояние записывается в структуру
*            MOUSE_STATE, описанную в файле sysp.h:
*
*               typedef struct _MOUSE_STATE_ {
*                     unsigned bottoms;
*                        unsigned x;
*                        unsigned y;
*                  } MOUSE_STATE;
*
*            Адрес структуры передается функции в качестве
*            параметра.
*
*.Proto      int ms_querp(MOUSE_STATE *state, int bottom);
*
*.Params     MOUSE_STATE *state - указатель на структуру,
*                                 описывающую состояние мыши.
*
*            int bottom - параметр определяет клавишу,
*                 при нажатии на которую необходимо
*                  запомнить состояние:
*                           0 - левая клавиша,
*                           1 - правая клавиша,
*                           2 - средняя клавиша.
*
*.Return     Функция возвращает количество нажатий на
*            заданную клавишу с момента последнего
*            вызова этой функции.
*
*.Sample     ms_samp3.c
**/
```

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include "sysp.h"

union REGS reg;

int ms_querp(MOUSE_STATE *state, int bottom) {

     reg.x.ax = 5;
     reg.x.bx = bottom;
     int86(0x33,&reg,&reg);

     state->bottoms = reg.x.ax;
     state->x       = reg.x.cx;
     state->y       = reg.x.dx;

     return(reg.x.bx);
}
```

3.3.7. Определить положение курсора при отпускании клавиши

На входе:	AX	=	0006h;
	BX	=	клавиша, при отпускании которой запоминается состояние мыши: 0 - левая; 1 - правая; 2 - средняя.
На выходе:	AX	=	состояние клавиш мыши: бит 0 = 1 - нажата левая клавиша; бит 1 = 1 - нажата правая клавиша; бит 2 = 1 - нажата средняя клавиша (для мыши системы Mouse Systems);
	BX	=	количество нажатий на заданную клавишу, обнуляется после вызова функции;
	CX	=	координата X (по горизонтали);
	DX	=	координата Y (по вертикали).

Эта функция возвращает программе состояние мыши, запомненное в момент отпускания клавиши, которая была заранее определена при вызове функции. Она также возвращает количество отпусканий заданной клавиши.

Функция для определения состояния мыши при отпускании заданной клавиши:

```
/**
*.Name       ms_querr
*.Title      Определение состояния мыши при отпускании клавиши
*
*.Descr      Эта функция определяет координаты курсора мыши
*            и состояние клавиш в момент отпускания заранее
*            заданной клавиши.
*            Определенное состояние записывается в структуру
*            MOUSE_STATE, описанную в файле sysp.h:
*
*               typedef struct _MOUSE_STATE_ {
*                     unsigned bottoms;
*                     unsigned x;
*                     unsigned y;
*               } MOUSE_STATE;
*
*            Адрес структуры передается функции в качестве
*            параметра.
*
*.Proto      int ms_querr(MOUSE_STATE *state, int bottom);
*
*.Params     MOUSE_STATE *state - указатель на структуру,
*                                 описывающую состояние мыши.
*
*            int bottom - параметр определяет клавишу,
*                 при отпускании которой необходимо
*                 запомнить состояние:
*                           0 - левая клавиша,
*                           1 - правая клавиша,
*                           2 - средняя клавиша.
*
*.Return        Функция возвращает количество отпусканий
*               заданной клавиши с момента последнего
*               вызова этой функции.
*
*.Sample        ms_samp3.c
**/

#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include "sysp.h"

union REGS reg;

int ms_querr(MOUSE_STATE *state, int bottom) {
```

```
    reg.x.ax = 6;
    reg.x.bx = bottom;
    int86(0x33,&reg,&reg);

    state->bottoms = reg.x.ax;
    state->x       = reg.x.cx;
    state->y       = reg.x.dx;

    return(reg.x.bx);
}
```

Приведем пример программы для определения и вывода на экран состояния мыши при нажатии и отпускании левой клавиши:

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

union REGS reg;
void main() {

    int botm, i;
    MOUSE_STATE state;
    unsigned old_videomode, new_videomode;
    char buf[20], *bufptr;

// Определяем текущий видеорежим

    reg.x.ax = 0x0f00;
    int86(0x10, &reg, &reg);
    old_videomode = reg.h.al;

// Устанавливаем новый видеорежим:

// Устанавливаем максимально допустимую длину строки

    buf[0] = 10;
    printf("\nВведите десятичный номер видеорежима: ");
    bufptr = cgets(buf);

// Преобразуем введенное число к формату int

    new_videomode = atoi(bufptr);

    reg.h.ah = 0;
    reg.h.al = new_videomode;
    int86(0x10, &reg, &reg);

// Инициализируем мышь, определяем количество клавиш

    if(!ms_init(&botm)) {
        printf("\nМышь не установлена");
        exit(-1);
    }
```

```
    printf("\nУстановлена мышь: ");
    switch (botm) {
      case 2:
              printf("двухклавишная"); break;
      case 3:
              printf("трехклавишная, системы Mouse Systems");
              break;
      case 0:
      default:
              printf("неизвестной системы"); break;
    }
    printf("\n\nСостояние мыши:\n\n");

// Включаем курсор

    ms_on();

    while(!kbhit()) {

// Определяем состояние мыши при нажатии на левую клавишу

          i = ms_querp(&state, 0);

// Если были нажатия на левую клавишу, выводим состояние мыши

          if(i != 0) {

// Перед выводом на экран отключаем курсор, затем включаем его
// снова.

               ms_off();
               printf("Нажатие:    %2d x:%5d y:%5d\n",
                    state.bottoms,
                    state.x,
                    state.y);
               ms_on();

          }

// Выводим состояние при отпускании клавиши

          i = ms_querr(&state, 0);
          if(i != 0) {

               ms_off();
               printf("Отпускание: %2d x:%5d y:%5d\n\n",
                    state.bottoms,
                    state.x,
                    state.y);
               ms_on();
          }
    }
    getch();

    ms_off();
```

```
    reg.h.ah = 0;
    reg.h.al = old_videomode;
    int86(0x10, &reg, &reg);
}
```

3.3.8. Задать диапазон движения курсора по горизонтали

На входе: AX = 0007h;
CX = минимальная координата X (по горизонтали);
DX = максимальная координата X.

На выходе: Регистры не используются.

Данная функция позволяет ограничить диапазон перемещений курсора мыши по горизонтали.

Вызов функции:

```
/**
*.Name       ms_rangx
*.Title      Задание диапазона перемещения курсора по горизонтали
*
*.Descr      Эта функция ограничивает область перемещения
*            курсора по горизонтали в пределах [xmin, xmax].
*
*.Proto      void ms_rangx(int xmin, int xmax)
*
*.Params     int xmin    - минимальная координата X курсора;
*            int xmax    - максимальная координата X курсора.
*
*.Return     Ничего
*
*.Sample     ms_samp4.c
**/

#include <dos.h>
#include <conio.h>

union REGS reg;

void ms_rangx(int xmin, int xmax) {

    reg.x.ax = 7;
    reg.x.cx = xmin;
    reg.x.dx = xmax;

    int86(0x33,&reg,&reg);
}
```

3.3.9. Задать диапазон движения курсора по вертикали

На входе: AX = 0008h;
CX = минимальная координата Y (по вертикали);
DX = максимальная координата Y.

На выходе: Регистры не используются.

Данная функция позволяет ограничить диапазон перемещений курсора мыши по вертикали.

Вызов функции:

```
/**
*.Name      ms_rangy
*.Title     Задание диапазона перемещения курсора по вертикали
*
*.Descr     Эта функция ограничивает область перемещения
*           курсора по вертикали в пределах [ymin, ymax].
*
*.Proto     void ms_rangy(int ymin, int ymax)
*
*.Params    int ymin   - минимальная координата Y курсора;
*           int ymax   - максимальная координата Y курсора.
*
*.Return    Ничего
*
*.Sample    ms_samp4.c
**/

#include <dos.h>
#include <conio.h>

union REGS reg;

void ms_rangy(int ymin, int ymax) {

     reg.x.ax = 8;
     reg.x.cx = ymin;
     reg.x.dx = ymax;

     int86(0x33,&reg,&reg);
}
```

Приведем текст программы, которая ограничивает диапазон перемещений курсора мыши по экрану:

```
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

void main() {

     int botm;

// Инициализируем мышь

     if(!ms_init(&botm)) {
          printf("\nМышь не установлена");
          exit(-1);
     }

// Включаем курсор и ожидаем нажатия на клавишу

     printf("\n\nКурсор мыши включен, для выключения"
               " нажмите любую клавишу");

     ms_on();

// Задаем границы, в которых должен перемещаться курсор

     ms_rangx(20, 100);
     ms_rangy(50, 100);

     getch();

// Выключаем курсор

     ms_off();
}
```

3.3.10. Задать форму курсора в графическом режиме

На входе: AX = 0009h;
BX = номер позиции точки-указателя графического курсора (от -16 до 16);
CX = номер строки точки-указателя (от -16 до 16);
ES:DX = указатель на битовое изображение курсора.

На выходе: Регистры не используются.

С помощью этой функции программа, работающая в графическом режиме, может изменить форму курсора мыши и положение точки внутри изображения курсора, координаты которой используются в качестве координат курсора остальными функциями.

Регистры ES:DX указывают на область длинной 64 байта. Эта область состоит из двух массивов длиной по 32 байта. Первый массив представляет из себя логическую маску размером 16x16 битов, которая накладывается на участок видеопамяти с использованием логической операции "И". Второй массив - тоже маска размером 16x16, но она накладывается с использованием логической операции "Исключающее ИЛИ", инвертируя отдельные точки изображения.

Номера позиции и строки точки-указателя, устанавливаемые по умолчанию, равны 0 (BX=CX=0). Это соответствует верхней левой точке в изображении курсора. Значения BX=CX=15 соответствуют нижней правой точке.

Приведем исходный текст функции для изменения формы курсора из программы, составленной на языке Си:

```
/**
*.Name          ms_gform
*.Title         Задание формы курсора в графическом режиме
*
*.Descr         Эта функция определяет форму курсора мыши
*               для графического режима. Дополнительно можно
*               задать положение точки-указателя, которая
*               соответствует координатам курсора.
*
*.Proto         void ms_gform(int xt, int yt, char _far *form)
*
*.Params        int xt - позиция точки указателя;
*               int yt - номер строки точки-указателя;
*               char *form - указатель на массив, описывающий
*               курсор.
*
*.Return        Ничего
*
*.Sample        ms_samp5.c
**/

#include <dos.h>
#include <conio.h>

union REGS reg;
struct SREGS segregs;

void ms_gform(int xt, int yt, char _far *form) {

     reg.x.ax = 9;
     reg.x.bx = xt;
     reg.x.cx = yt;
```

```
    reg.x.dx = FP_OFF(form);
    segregs.es = FP_SEG(form);
    int86x(0x33,&reg,&reg,&segregs);
}
```

Мы подготовили пример программы, изменяющей форму курсора в графическом режиме:

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

union REGS reg;

char form[64] = {

// Массив маски по "И"

    255,255, 255,255, 255,255, 255,255, 255,255,
          255,255, 255,255, 255,255,
    255,255, 255,255, 255,255, 255,255, 255,255,
         255,255, 255,255, 255,255,

// Массив маски по "Исключающее ИЛИ"

    127,254, 127,254, 127,254, 127,254, 127,254,
          127,254, 127,254, 0,0,
    0,0, 127,254, 127,254, 127,254, 127,254, 127,254,
          127,254, 127,254,
    };


void main() {

    int botm, i;
    MOUSE_STATE state;
    unsigned old_videomode, new_videomode;
    char buf[20], *bufptr;

// Определяем текущий видеорежим

    reg.x.ax = 0x0f00;
    int86(0x10, &reg, &reg);
    old_videomode = reg.h.al;

// Устанавливаем новый видеорежим:

// Устанавливаем максимально допустимую длину строки

    buf[0] = 10;
    printf("\nВведите десятичный номер видеорежима: ");
    bufptr = cgets(buf);
```

```
// Преобразуем введенное число к формату int
     new_videomode = atoi(bufptr);
     reg.h.ah = 0;
     reg.h.al = new_videomode;
     int86(0x10, &reg, &reg);
// Инициализируем мышь
     if(!ms_init(&botm)) {
          printf("\nМышь не установлена");
          exit(-1);
     }
// Задаем новую форму для курсора мыши
     ms_gform(0,0, &form[0]);
// Включаем курсор
     ms_on();
     getch();
     ms_off();
     reg.h.ah = 0;
     reg.h.al = old_videomode;
     int86(0x10, &reg, &reg);
}
```

3.3.11. Задать форму курсора в текстовом режиме

На входе: AX = 000Ah;
BX = тип курсора:
0 - определяемый программно;
1 - определяемый аппаратно;
CX = маска экрана (для BX=0) или начальная строка курсора (для BX=1);
DX = маска курсора (для BX=0) или конечная строка курсора (для BX=1).

На выходе: Регистры не используются.

С помощью этой функции программа может изменять форму курсора мыши в текстовом режиме.

В зависимости от содержимого регистра BX драйвер мыши использует курсор, определяемый аппаратными средствами, либо

курсор, определяемый программно. По умолчанию используется "программный курсор", который отображается в виде символа с инвертированным значением атрибута. Курсор, сформированный аппаратными средствами, выглядит аналогично обычному текстовому курсору, его форма - прямоугольник. Размер этого прямоугольника можно задавать при помощи регистров CX и DX.

Для курсора, определяемого программно, вначале выполняется операция логического "И" над содержимым видеопамяти в том месте, куда указывает курсор, и маской экрана. Затем выполняется операция "Исключающее ИЛИ" с маской курсора.

Младший байт масок соответствует ASCII-коду символа, старший - это байт атрибута символа.

Значения, используемые по умолчанию, - BX=7700h, CX=FFFFh.

Если Вам надо изменить цвет курсора, не меняя его форму, задайте CX=00FFh, BX=xx00h, где 'xx' определяет цвет (см. описание формата байта атрибутов в третьем томе книги).

Приведем функцию для изменения формы курсора в текстовом примере и программу, создающую курсор в виде вертикальной стрелки, направленной вверх на синем фоне:

```
/**
*.Name          ms_tform
*.Title         Задание формы курсора в текстовом режиме
*
*.Descr         Эта функция определяет форму курсора мыши для
*               текстового режима.
*
*.Proto         void ms_tform(int type, int mask1, int mask2)
*
*.Params        int type - тип курсора:
*                       0 - программный, 1 - аппаратный;
*               int mask1 - AND-маска экрана
*                       или первая строка аппаратного курсора
*               int mask2 - XOR-маска курсора
*                       или последняя строка аппаратного курсора
*
*.Return        Ничего
*
*.Sample        ms_samp6.c
**/

#include <dos.h>
#include <conio.h>
```

```
union REGS reg;

void ms_tform(int type, int mask1, int mask2) {

     reg.x.ax = 0xA;
     reg.x.bx = type;
     reg.x.cx = mask1;
     reg.x.dx = mask2;

     int86(0x33,&reg,&reg);
}
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

union REGS reg;

void main() {

     int botm, i;
     MOUSE_STATE state;

// Инициализируем мышь

     if(!ms_init(&botm)) {
          printf("\nМышь не установлена");
          exit(-1);
     }

// Задаем новую форму для курсора мыши

     ms_tform(0, 0, 0x1418);

// Включаем курсор

     ms_on();

     getch();

     ms_off();
}
```

3.3.12. Определить содержимое счетчиков перемещения

На входе: AX = 000Bh.

На выходе: CX = перемещение по горизонтали с момента последнего вызова функции;

DX = перемещение по вертикали с момента последнего вызова функции.

Функция позволяет определить относительное перемещение мыши с момента последнего вызова этой функции. Результат возвращается в указанных выше регистрах. Для измерения перемещения используется единица mickey - "мики". Один мик соответствует 0,005 дюйма (т. е. 1/200 дюйма).

Отрицательные значения перемещения означают соответственно движение влево и вверх, положительные - вправо и вниз.

Для преобразования миков в пиксели (точки экрана) можно использовать функцию 1Bh, которая будет описана позже.

Функция для определения относительного перемещения:

```
/**
*.Name       ms_querm
*.Title      Определение показания счетчика перемещений
*
*.Descr      Эта функция определяет относительное перемещение
*            мыши с момента последнего вызова функции.
*
*            Определенное состояние записывается в структуру
*            MOUSE_STATE, описанную в файле sysp.h:
*
*               typedef struct _MOUSE_STATE_ {
*                     unsigned bottoms;
*                     unsigned x;
*                     unsigned y;
*               } MOUSE_STATE;
*
*            Адрес структуры передается функции в качестве
*            параметра. Перемещение определяется в миках -
*            1/200 долях дюйма.
*
*.Proto          MOUSE_STATE *ms_querm(MOUSE_STATE *state);
*
*.Params         MOUSE_STATE *state - указатель на структуру,
*               описывающую состояние мыши.
*
*.Return         Функция возвращает значение своего параметра.
*
*.Sample         ms_samp7.c
**/

#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include "sysp.h"

union REGS reg;
```

```
MOUSE_STATE *ms_querm(MOUSE_STATE *state) {
     reg.x.ax = 0x8;
     int86(0x33,&reg,&reg);
     state->bottoms = 0;
     state->x       = reg.x.cx;
     state->y       = reg.x.dx;
     return(state);
}
```

Пример программы, постоянно вызывающей эту функцию и отображающей на экране относительное перемещение мыши:

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include "sysp.h"
union REGS reg;
void main() {
     int botm, i;
     MOUSE_STATE state;
     unsigned old_videomode, new_videomode;
     char buf[20], *bufptr;

// Определяем текущий видеорежим
     reg.x.ax = 0x0f00;
     int86(0x10, &reg, &reg);
     old_videomode = reg.h.al;
// Устанавливаем новый видеорежим:
// Устанавливаем максимально допустимую длину строки
     buf[0] = 10;
     printf("\nВведите десятичный номер видеорежима: ");
     bufptr = cgets(buf);
// Преобразуем введенное число к формату int
     new_videomode = atoi(bufptr);
     reg.h.ah = 0;
     reg.h.al = new_videomode;
     int86(0x10, &reg, &reg);
// Инициализируем мышь, определяем количество клавиш
     if(!ms_init(&botm)) {
          printf("\nМышь не установлена");
          exit(-1);
     }
     printf("\n\nСостояние мыши:\n\n");
```

```
// Включаем курсор
     ms_on();
     while(!kbhit()) {
          ms_querm(&state);
          printf("x:%5d y:%5d",
               state.x,
               state.y);
          for(i=0;i<15;i++) printf("\b");
     }
     getch();
     ms_off();
     reg.h.ah = 0;
     reg.h.al = old_videomode;
     int86(0x10, &reg, &reg);
}
```

3.3.13. Установить драйвер событий

На входе: AX = 000Ch;
CX = маска вызова:
бит 0 - вызов при перемещении мыши;
бит 1 - вызов при нажатии левой клавиши;
бит 2 - вызов при отпускании левой клавиши;
бит 3 - вызов при нажатии правой клавиши;
бит 4 - вызов при отпускании правой клавиши;
бит 5 - вызов при нажатии средней клавиши;
бит 6 - вызов при отпускании средней клавиши;
7Fh - вызов при любом событии;
00h - отключение драйвера событий;
ES:DX = адрес (дальний) подключаемого драйвера событий.

На выходе: Регистры не используются.

Функция позволяет программе создать свой собственный драйвер (обработчик) событий, связанных с перемещением мыши и нажатием/отпусканием клавиш мыши.

Адрес подготовленной программы-драйвера передается при вызове функции в регистровой паре ES:DX. Драйвер должен быть оформлен в виде дальней процедуры, завершающейся командой дальнего возврата RETF. Когда драйвер получает управление, в регистрах процессора содержатся следующие значения:

AX	Маска вызова, такая же, как и при вызове функции 0Ch.
BX	Состояние клавиш мыши: бит 0 - левая клавиша; бит 1 - правая клавиша; бит 2 - средняя клавиша.
CX	Горизонтальная координата курсора мыши.
DX	Вертикальная координата курсора мыши.
SI	Относительное перемещение мыши по горизонтали в миках.
DI	Относительное перемещение мыши по вертикали в миках.
DS	Сегмент данных драйвера мыши.

Так как регистр DS при вызове драйвера событий содержит сегмент данных драйвера мыши, Ваш драйвер событий должен позаботиться о правильной установке этого регистра. Ваш драйвер событий не обязан сохранять и восстанавливать содержимое регистра DS и других регистров процессора.

Отметим, что, если Вам необходимо отключить драйвер, выполните повторный вызов функции 0Ch, записав в регистр CX нулевое значение. Если Ваша программа, устанавливающая собственный драйвер событий, завершает свою работу и передает управление MS-DOS, предварительно она обязательно должна отключить драйвер событий.

Приведем функцию, которую мы разработали для подключения драйвера событий:

```
/**
*.Name          ms_seth
*.Title         Установка драйвера событий
*
*.Descr         Эта функция выполняет установку драйвера событий.
*
*.Proto         void ms_seth(int mask, void (far *hand)())
*
*.Params        int mask - маска событий;
*               void (far *hand)() - адрес драйвера событий
*
*.Return        Ничего
*
*.Sample        ms_samp8.c
**/

#include <dos.h>
#include <conio.h>

union REGS reg;
struct SREGS segregs;

void ms_seth(int mask, void (far *hand)()) {

     reg.x.ax = 0x14;
     reg.x.cx = mask;
     reg.x.dx = FP_OFF(hand);
     segregs.es = FP_SEG(hand);

     int86x(0x33,&reg,&reg,&segregs);
}
```

Составление программы драйвера событий имеет некоторые особенности. Драйвер событий вызывается не из программы пользователя, а из драйвера мыши. При этом сегментный регистр DS будет указывать на сегмент данных драйвера мыши, а не на сегмент данных Вашей программы.

Мы подготовили образец драйвера событий, использовав язык ассемблера:

```
;**
;.Name          ms_handl
;.Title         Драйвер событий
;
;.Descr         Драйвер событий вызывается драйвером мыши,
;               когда происходит какое-нибудь событие из числа
;               заданных при установке драйвера событий.
;               Функция не должна вызываться из программы
;               пользователя, ее вызывает только драйвер мыши.
;
```

```
;.Proto         void far ms_handl(void);
;
;.Params        Не используются
;
;.Return        Ничего
;
;.Sample        ms_samp8.c
;**

     DOSSEG
DGROUP  GROUP   _DATA

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA   ENDS

_TEXT   SEGMENT WORD PUBLIC 'CODE'
     ASSUME  cs:_TEXT, ds:DGROUP, ss:DGROUP

; Флаг вызова драйвера событий

extrn   _ms_flag:word

; Внешние переменные для записи содержимого регистров

extrn   _ms_bx:word
extrn   _ms_cx:word
extrn   _ms_dx:word
extrn   _ms_si:word
extrn   _ms_di:word
extrn   _ms_ds:word

public  _ms_handl

_ms_handl   proc far

     mov     _ms_ds, ds

; Так как на входе в драйвер событий регистр DS указывает на
; сегмент данных драйвера мыши, устанавливаем его на сегмент
; данных программы;

     push    ax
     mov     ax, DGROUP
     mov     ds, ax
     pop     ax

     mov     _ms_bx, bx
     mov     _ms_cx, cx
     mov     _ms_dx, dx
     mov     _ms_si, si
     mov     _ms_di, di

; Устанавливаем флаг вызова драйвера в единицу, сигнализируя
; программе о том, что произошло событие.
```

```
    mov     _ms_flag, 1
    ret
_ms_handl   endp
_TEXT   ENDS
    END
```

Этот драйвер при вызове устанавливает глобальную переменную _ms_flag в единицу, затем переписывает содержимое всех нужных регистров в соответствующие глобальные переменные. Программа, установив драйвер событий и сбросив флаг _ms_flag, может выполнять какие-либо действия (например, вывод на экран движущегося изображения), постоянно проверяя флаг _ms_flag. Как только произойдет какое-либо событие (нажатие или отпускание клавиши мыши, перемещение мыши), драйвер событий установит флаг в единицу. Программа при этом может узнать состояние мыши, прочитав содержимое глобальных переменных _ms_bx, _ms_dx, и т. д.

Этот простейший вариант использования драйвера событий иллюстрируется следующей программой:

```
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

extern void far ms_handl(void);

// Флаг драйвера событий. При вызове драйвер событий
// запишет в эту переменную значение 1.

    unsigned ms_flag;

// Область для содержимого регистров на входе
// в драйвер событий.

    unsigned ms_bx;
    unsigned ms_cx;
    unsigned ms_dx;
    unsigned ms_si;
    unsigned ms_di;
    unsigned ms_ds;

    int botm;

main () {
```

```
     ms_flag=0;
// Инициализируем мышь, определяем количество клавиш

     if(!ms_init(&botm)) {
          printf("\nМышь не установлена");
          exit(-1);
     }

// Подключаем драйвер событий, устанавливаем маску таким
образом,
// чтобы драйвер вызывался при нажатии на левую или правую
// клавиши мыши.

     ms_seth(2 | 8, ms_handl);

// Включаем курсор

     ms_on();

// Ожидаем вызова драйвера событий.

     for(;;) {
          if(ms_flag) {
               ms_off();

               printf("\nСостояние регистров "
                         "на входе драйвера:"
                    "\nms_bx: %0X"
                    "\nms_cx: %0X"
                    "\nms_dx: %0X"
                    "\nms_si: %0X"
                    "\nms_di: %0X"
                    "\nms_ds: %0X",
                    ms_bx,
                    ms_cx,
                    ms_dx,
                    ms_si,
                    ms_di,
                    ms_ds);

               printf("\nНажмите любую клавишу...");
               getch();
               exit(0);
          }
     }
}
```

Драйвер событий может также организовать очередь событий, записывая в эту очередь состояние мыши на момент появления события и время появления события.

Прикладная программа будет затем извлекать события из очереди и анализировать их.

3.3.14. Включить эмуляцию светового пера

На входе: AX = 000Dh.

На выходе: Регистры не используются.

Если Ваша программа использует световое перо (например, она написана на языке Бейсик и вызывает функцию PEN), Вы можете заменить световое перо на мышь. После включения режима эмуляции драйвер запоминает координаты курсора при нажатии на клавиши мыши. Эти координаты могут быть впоследствии считаны функцией PEN или функцией 04h прерывания INT 10h, предназначенной для работы со световым пером.

3.3.15. Выключить эмуляцию светового пера

На входе: AX = 000Eh.

На выходе: Регистры не используются.

Эта функция выключает режим эмуляции светового пера.

3.3.16. Задать скорость перемещения курсора мыши

На входе:
- AX = 000Fh;
- CX = количество миков на 8 точек по горизонтали;
- DX = количество миков на 8 точек по вертикали.

На выходе: Регистры не используются.

Функция определяет "чувствительность" мыши к перемещению по поверхности стола, т. е. устанавливает соответствие между величиной перемещения мыши по столу и величиной перемещения курсора мыши по экрану.

При инициализации драйвера мыши используются следующие значения: CX=8, DX=16.

3.3.17. Установить область исключения для курсора

На входе: AX = 0010h;

CX, DX = координаты (X, Y) верхнего левого угла области исключения;
SI, DI = координаты (X, Y) нижнего правого угла области исключения.

На выходе: Регистры не используются.

Функция позволяет задать на экране прямоугольную область, в которой автоматически выключается изображение курсора мыши, - область исключения. Эта область отменяется функциями 01h (включить курсор мыши) и 00h (инициализация).

Оператор может поместить курсор мыши в область исключения, при этом изображение курсора пропадет.

Основное назначение этой функции - предоставить программе возможность изменять содержимое области экрана не выключая изображение курсора. Недостаток функции - можно "потерять" курсор мыши, если он случайно окажется в области исключения.

3.3.18. Задать увеличенный графический курсор (PC MOUSE)

На входе: AX = 0012h;
BH = ширина курсора в словах;
CH = количество строк в изображении курсора;
BL = номер позиции точки-указателя графического курсора (от -16 до 16);
CL = номер строки точки-указателя (от -16 до 16);
ES:DX = указатель на битовое изображение курсора.

На выходе: Регистры не используются.

Эта функция позволяет задать увеличенный по размеру курсор мыши, но она определена только для мыши системы PC MOUSE.

3.3.19. Определить порог удвоения скорости

На входе: AX = 0013h.

На выходе: DX = значение порога удвоения, мики в секунду.

Если Вы перемещаете мышь со скоростью, превышающей порог удвоения, заданный функцией 13h, аппаратура мыши удваивает величину перемещения. Таким образом, используя медленное перемещение мыши, Вы можете точно устанавливать курсор на требуемый элемент изображения. Если Вам необходимо переместить курсор на значительное расстояние по экрану, Вы можете увеличить скорость перемещения мыши.

При инициализации устанавливается значение порога, равное 64 микам в секунду (1/3 дюйма в секунду). Если Вам надо установить это значение, Вы можете при вызове функции 13h задать DX=0.

3.3.20. Заменить драйвер событий

На входе: AX = 0014h;
CX = маска вызова:
бит 0 - вызов при перемещении мыши;
бит 1 - вызов при нажатии левой клавиши;
бит 2 - вызов при отпускании левой клавиши;
бит 3 - вызов при нажатии правой клавиши;
бит 4 - вызов при отпускании правой клавиши;
бит 5 - вызов при нажатии средней клавиши;
бит 6 - вызов при отпускании средней клавиши;
7Fh - вызов при любом событии;
00h - отключение драйвера событий;
ES:DX = адрес (дальний) подключаемого драйвера событий.

На выходе: CX = маска предыдущего драйвера событий;

ES:DX = адрес предыдущего драйвера событий (т. е. адрес заменяемого драйвера событий).

Функция аналогична функции 0Ch, однако ее основное назначение - временная замена драйвера событий. Например, подпрограмма в начале своей работы может установить свой драйвер событий, а перед завершением - активизировать драйвер, использовавшийся ранее.

3.3.21. Определить размер буфера состояния драйвера

На входе: AX = 0015h.

На выходе: BX = размер буфера, требующийся для хранения состояния драйвера мыши.

Если Вам требуется временно сохранить состояние драйвера мыши, а затем восстановить его, Вы можете воспользоваться специально предназначенными для этого функциями 16h и 17h. Для этих функций требуется буфер, в котором будет храниться состояние драйвера. Размер буфера можно определить с помощью функции 15h.

Когда может потребоваться запоминание и восстановление состояния драйвера? Например, при использовании мыши резидентными (TSR) программами желательно сохранить состояние драйвера перед началом работы TSR-программы и восстановить его перед завершением работы TSR-программы.

3.3.22. Сохранить состояние драйвера

На входе: AX = 0016h;
ES:DX = адрес буфера для записи состояния драйвера.

На выходе: Не используются.

Функция позволяет записать состояние драйвера в буфер, размер которого должен быть определен с помощью функции 15h.

3.3.23. Восстановить состояние драйвера

На входе: AX = 0017h;
ES:DX = адрес буфера, содержащего состояние драйвера.

На выходе: Не используются.

Функция позволяет восстановить состояние драйвера из буфера, в который оно было записано при помощи функции 16h.

3.3.24. Установить альтернативный драйвер событий

На входе: AX = 0018h;
CX = маска вызова:
бит 0 - вызов при перемещении мыши;
бит 1 - вызов при нажатии левой клавиши;
бит 2 - вызов при отпускании левой клавиши;
бит 3 - вызов при нажатии правой клавиши;
бит 4 - вызов при отпускании правой клавиши;
бит 5 - вызов при одновременном нажатии клавиши мыши и клавиши SHIFT на клавиатуре;
бит 6 - вызов при одновременном нажатии клавиши мыши и клавиши CTRL на клавиатуре;
бит 7 - вызов при одновременном нажатии клавиши мыши и клавиши ALT на клавиатуре;
7Fh - вызов при любом событии;
00h - отключение драйвера событий;
ES:DX = адрес (дальний) подключаемого драйвера событий.

На выходе: AX = результат установки:
0018h - драйвер успешно установлен;
FFFFh - ошибка при установке драйвера.

По сравнению с функцией 0Ch эта функция обеспечивает дополнительные возможности:

- проверка состояния клавиш SHIFT, CTRL, ALT во время нажатия на клавиши мыши;
- возможность одновременной установки до трех драйверов событий, каждый из которых использует свою маску событий, задаваемых в регистре CX.

При попытке установить два драйвера с одной и той же маской событий функция возвращает в регистре AX код ошибки FFFFh. В этом случае можно использовать функцию 19h для получения адреса предыдущего установленного драйвера событий, отключить его и повторить подключение своего драйвера.

Вы можете использовать функцию 18h для отключения драйвера событий, если укажете в регистрах ES:DX его адрес и зададите в регистре CX значение маски, равное нулю.

3.3.25. Получить адрес альтернативного драйвера событий

На входе:	AX	=	0019h;
	CX	=	маска событий, для которой требуется получить адрес драйвера (формат маски соответствует функции 18h).
На выходе:	CX	=	маска событий или 0000h, если заданной маске не соответствует ни один установленный драйвер событий;
	ES:DX	=	адрес драйвера событий, использующий заданную маску событий.

Эта функция предназначена для получения адреса драйвера событий с заданной маской событий. Получив адрес, Вы можете установить новый драйвер, использующий эту же маску.

3.3.26. Установить чувствительность мыши

На входе:	AX	=	001Ah;
	BX	=	горизонтальная чувствительность, мики на точку (пиксель);

CX = вертикальная чувствительность, мики на точку (пиксель);
DX = значение порога удвоения, мики в секунду.

На выходе: Не используются.

Эта функция является комбинацией функций 0Fh и 13h. Она позволяет одновременно устанавливать чувствительность мыши и порог удвоения скорости.

3.3.27. Определить чувствительность мыши

На входе: AX = 001Bh.

На выходе: BX = горизонтальная чувствительность, мики на точку (пиксель);
CX = вертикальная чувствительность, мики на точку (пиксель);
DX = значение порога удвоения, мики в секунду.

Функция позволяет определить текущие значения для чувствительности мыши и порога удвоения.

3.3.28. Установить частоту прерываний для Inport Mouse

На входе: AX = 001Ch
BX = код скорости прерываний:
1 - нет прерываний;
2 - 30 прерываний в секунду;
4 - 50 прерываний в секунду;
8 - 100 прерываний в секунду;
16 - 200 прерываний в секунду.

На выходе: Не используются.

Мышь периодически вырабатывает сигнал прерывания, по которому драйвер считывает текущее состояние мыши. С помощью функции 1Ch Вы можете изменять частоту появления прерываний, но только для мыши системы Inport Mouse (Вы можете определить тип используемой мыши с помощью функции 24h).

Если используется большая частота прерываний, возрастает точность определения состояния мыши, но уменьшается общая производительность системы.

3.3.29. Установить номер видеостраницы

На входе: AX = 001Dh;
BX = номер видеостраницы.

На выходе: Не используются.

Данная функция задает номер видеостраницы, на которой будет отображаться курсор мыши. По умолчанию для отображения используется страница 0.

3.3.30. Определить номер видеостраницы

На входе: AX = 001Eh.

На выходе: BX = номер видеостраницы.

Функция возвращает номер видеостраницы, на которой в настоящее время отображается курсор мыши.

3.3.31. Отключить драйвер мыши

На входе: AX = 001Fh.

На выходе: AX = результат выполнения:
001Fh - драйвер отключен;
FFFFh - отключение невозможно;
ES:DX = вектор предыдущего драйвера мыши.

После вызова функции драйвер мыши полностью отключается. Вектор прерывания INT 33h остается определенным, однако выполняется лишь одна функция прерывания INT 33h - 21h (программный сброс мыши). Функцию 1Fh удобно использовать для временной замены драйвера на собственную систему обслуживания мыши. Сначала Вы отключаете драйвер функцией 1Fh, запоминая вектор предыдущего драйвера, возвращаемого в регистрах ES:DX. Затем устанавливаете собственную систему обслуживания. Впоследствии Вы восстанавливаете значение этого вектора.

3.3.32. Восстановить драйвер мыши

На входе: AX = 0020h.

На выходе: Не используются.

Функция восстанавливает связь между мышью и драйвером, отключенную вызовом функции 1Fh.

3.3.33. Сбросить драйвер мыши

На входе: AX = 0021h.

На выходе: AX = результат:
0021h - драйвер сброшен успешно;
FFFFh - невозможно сбросить драйвер (например, нет драйвера);
BX = количество клавиш на корпусе мыши.

Функция аналогична функции 00h, но она не выполняет аппаратного сброса оборудования мыши.

3.3.34. Определить тип мыши

На входе: AX = 0024h.

На выходе: BH = верхний (major) номер версии драйвера;
BL = нижний (minor) номер версии драйвера;
CH = тип мыши:
1 - Bus Mouse;
2 - Serial Mouse;
3 - Inport Mouse;
4 - PS/2 Mouse;
5 - HP Mouse;
CL = номер используемого прерывания (IRQ):
0 - IBM PS/2;
2,3,4,5,7 - IBM PC.

Эта функция дает информацию о типе используемой мыши, версии драйвера мыши и об используемом номере прерывания.

Глава 4

ЧАСЫ РЕАЛЬНОГО ВРЕМЕНИ

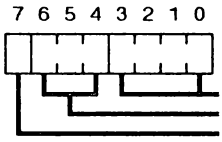
Компьютеры IBM AT и PS/2 оснащены часами реального времени. Эти часы питаются от аккумулятора, поэтому их показания не пропадают при выключении компьютера.

Доступ к часам реального времени возможен либо через ячейки КМОП-памяти, либо через специальные функции BIOS (что более предпочтительно с точки зрения независимости работы программы от особенностей аппаратуры).

Использование регистров КМОП-памяти часами реального времени приведено в таблице:

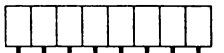
Регистр	*Назначение*
0	Счетчик секунд
1	Регистр секунд будильника
2	Счетчик минут
3	Регистр минут будильника
4	Счетчик часов
5	Регистр часов будильника
6	Счетчик дней недели (1 - воскресенье)
7	Счетчик дней месяца
8	Счетчик месяцев
9	Счетчик лет (последние две цифры текущего года)

0aH Регистр состояния A



Переключатель скорости (установлен в 0110)
22-разрядный делитель (установлен в 010)
Флаг обновления, 0 означает готовность данных для чтения

```
0bH     Регистр состояния B

 7 6 5 4 3 2 1 0
```



```
                1 - использование летнего времени
                (daylight savings enable);
                0 - стандартное время (установлен в 0)
              12 или 24-часовой режим.
              0 - 12-часовой режим
              (установлен в 1)
            Режим данных BCD.
            1 - двоичный, 0 - BCD.
            (установлен в 0)
          Разрешение прямоугольной волны.
          1 - включение прямоугольной волны.
          (установлен в 0)
        Разрешение прерывания по окончанию
        изменения данных (установлен в 0)
      Разрешение прерывания будильника
      (установлен в 0)
    Разрешение периодических прерываний
    (установлен в 0)
  Флаг обновления, 0 означает готовность
  данных для чтения КМОП-памяти

0cH     Регистр состояния C.
        Биты состояния прерывания, их можно только читать

0dH     Регистр состояния D.
        Если бит 7 равен 0, это означает, что разрядился
        аккумулятор, питающий КМОП-память
```

Часы реального времени вырабатывают аппаратное прерывание IRQ8, которому соответствует прерывание с номером 70h. Это прерывание может вырабатываться по трем причинам:

- Прерывание по окончании изменения данных. Вырабатывается при установленном в 1 бите 4 регистра состояния B после каждого обновления регистров часов.
- Прерывание будильника вырабатывается при совпадении регистров часов и регистров будильника и при установленном в 1 бите 5 регистра состояний B.
- Периодическое прерывание вырабатывается с интервалом примерно 1 миллисекунда при установленном в 1 бите 6 регистра состояний B.

При срабатывании будильника BIOS вырабатывает прерывание INT 4Ah. Программа может подготовить собственный обработчик для этого прерывания.

Для работы с часами реального времени Вы можете обращаться непосредственно к перечисленным выше ячейкам КМОП-памяти, используя порты 70h и 71h. Однако лучше всего воспользоваться функциями 2 - 7 прерывания 1Ah, описанными ниже.

4.1. Прочитать показания часов реального времени

На входе:	AH	=	02h.
На выходе:	CH	=	часы в BCD-формате (например, 13h означает 13 часов);
	CL	=	минуты в BCD-формате;
	DH	=	секунды в BCD-формате;
	CF	=	CY = 1, если часы реального времени не установлены.

4.2. Установить часы реального времени

На входе:	AH	=	03h;
	CH	=	часы в BCD-формате (например, 13h означает 13 часов);
	CL	=	минуты в BCD-формате;
	DH	=	секунды в BCD-формате;
	DL	=	1, если необходимо использовать летнее время (daylight savings time option).

На выходе: Не используются.

4.3. Прочитать дату из часов реального времени

На входе:	AH	=	04h.
На выходе:	CH	=	столетие в BCD-формате ;
	CL	=	год в BCD-формате (например, CX=1991h означает 1991 год);
	DH	=	месяц в BCD-формате;
	DL	=	число в BCD-формате;

	CF	=	CY = 1, если часы реального времени не установлены.

4.4. Установить дату в часах реального времени

На входе:	AH	=	05h;
	CH	=	столетие в BCD-формате ;
	CL	=	год в BCD-формате (например, CX=1991h означает 1991 год);
	DH	=	месяц в BCD-формате;
	DL	=	число в BCD-формате;
На выходе:	не используются.		

4.5. Установить будильник

На входе:	AH	=	06h;
	CH	=	часы в BCD-формате;
	CL	=	минуты в BCD-формате;
	DH	=	секунды в BCD-формате.
На выходе:	CF	=	CY = 1, если часы реального времени не установлены.

Функция позволяет установить будильник на заданное время. Когда будильник "зазвенит", будет вызвано прерывание INT 4Ah (прерывание вызывают модули BIOS после прихода аппаратного прерывания от часов реального времени IRQ8, т. е. прерывания с номером 70h). Программа, использующая функцию будильника, должна подготовить обработчик прерывания INT 4Ah, завершающий свою работу выполнением команды IRET.

Программа может установить только один будильник.

4.6. Сброс будильника

На входе: AH = 07h.

На выходе: Не используются.

Эта функция позволяет сбросить будильник, например для того, чтобы установить его заново на другое время.

4.7. Использование часов реального времени

Вы можете использовать часы реального времени для решения двух задач. Во-первых, часы позволяют определить текущую дату и время с точностью до секунды. Во-вторых, будильник можно использовать для выполнения каких-либо действий в заданное время или периодически.

Так как установленное время срабатывания будильника хранится в КМОП-памяти, питающейся от аккумулятора, будильник не будет сброшен при случайном выключении компьютера.

Для работы с часами реального времени мы подготовили следующую функцию:

```
/**
*.Name      timer
*.Title     Работа с часами реального времени
*
*.Descr     Эта функция предназначена для обслуживания
*           системных часов реального времени через
*           прерывание INT 1Ah.
*
*.Proto     int timer(char fn, SYSTIMER *tm)
*
*.Params    char     fn - выполняемая функция:
*
*           RTC_GET_TIME      - прочитать показания часов,
*           RTC_SET_TIME      - установить часы;
*           RTC_GET_DATE      - прочитать дату;
*           RTC_SET_DATE      - установить дату;
*           RTC_SET_ALARM     - установить будильник;
*           RTC_CLEAR_ALARM   - сбросить будильник.
*
*              Все эти константы описаны в файле sysp.h
*
*           SYSTIMER tm - структура, содержащая данные
*                         для установки часов или
*                         показания часов:
*
*           typedef struct _SYSTIMER_ {
*
*              char hour;      // часы
*              char min;       // минуты
*              char sec;       // секунды
*              unsigned year;  // год
*              char month;     // месяц
*              char day;       // число
*              char daylight_savings; // флаг
```

```
*                       // использование
*                       // летнего времени
*                       // (для включения режима
*                       // должен быть равен 1)
*
*           } SYSTIMER;
*
*.Return    0   - успешное выполнение функции;
*           -1  - часы реального времени отсутствуют
*                 в компьютере;
*
*.Sample    setalarm.c
**/

#include <stdio.h>
#include <dos.h>
#include "sysp.h"

union REGS reg;

int timer(char fn, SYSTIMER *tm) {

     reg.h.ah = fn;

     switch (fn) {

          case RTC_SET_TIME:

               reg.h.ch = tm->hour;
               reg.h.cl = tm->min;
               reg.h.dh = tm->sec;
               reg.h.dl = tm->daylight_savings;

               break;

          case RTC_SET_DATE:

               reg.x.cx = tm->year;
               reg.h.dh = tm->month;
               reg.h.dl = tm->day;

               break;

          case RTC_SET_ALARM:

               reg.h.ch = tm->hour;
               reg.h.cl = tm->min;
               reg.h.dh = tm->sec;

               break;

     }

     int86(0x1a,&reg,&reg);

     if(reg.x.cflag == 1) return(-1);
```

```
    switch (fn) {
        case RTC_GET_TIME:
            tm->hour = reg.h.ch;
            tm->min = reg.h.cl;
            tm->sec = reg.h.dh;
            break;
        case RTC_GET_DATE:
            tm->year = reg.x.cx;
            tm->month = reg.h.dh;
            tm->day = reg.h.dl;
            break;
    }
    return(0);
}
```

Эта функция выполняет все виды обслуживания часов реального времени, которые поддерживаются BIOS.

Для иллюстрации основных приемов работы с часами мы подготовили программу, которая выводит на экран текущую дату и время. Затем устанавливается будильник. Он должен сработать через одну минуту и подать звуковой сигнал.

Перед установкой будильника программа подключает свой обработчик прерывания 4Ah. Это прерывание вызывается при срабатывании будильника. Перед завершением работы программа сбрасывает будильник и восстанавливает вектор прерывания 4Ah.

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include "sysp.h"

// Выключаем проверку стека и указателей

#pragma check_stack( off )
#pragma check_pointer( off )

// Макро для выдачи звукового сигнала

#define BEEP() _asm { \
    _asm mov bx,0 \
    _asm mov ax, 0E07h \
    _asm int 10h \
}

void main(void);
```

```
// Описание программы-обработчика прерывания
// будильника

void _interrupt _far alarm(void);

// Переменная для хранения старого
// вектора будильника

void (_interrupt _far *old_4a)(void);

void main(void) {

char *month_to_text[] = {

      "январь",
      "февраль",
      "март",
      "апрель"
      "май",
      "июнь",
      "июль",
      "август",
      "сентябрь",
      "октябрь",
      "ноябрь",
      "декабрь"
};

SYSTIMER tmr;

// Определяем текущие дату и время

      timer(RTC_GET_DATE, &tmr);
      timer(RTC_GET_TIME, &tmr);

// Выводим дату и время на экран

      printf("\nСейчас %d год, %s, %d число
            "\n",
            bcd2bin(&(tmr.year)),
            month_to_text[bcd1bin(&(tmr.month)) - 1],
            bcd1bin(&(tmr.day)));

      printf("\nВремя - %02.2d:%02.2d:%02.2d"
            "\n",
            bcd1bin(&(tmr.hour)),
            bcd1bin(&(tmr.min)),
            bcd1bin(&(tmr.sec)));

// Для установки будильника увеличиваем
// счетчик минут на единицу. Для упрощения
// программы мы не проверяем счетчик на
// переполнение, поэтому если текущее
// значение счетчика минут равно 59,
// будильник не сработает. Вы можете сами
```

```
// немного усовершенствовать программу для
// проверки переполнения.

     bin1bcd(bcd1bin(&(tmr.min)) + 1,
                                &(tmr.min));

// Выводим на экран время, когда сработает
// будильник.

     printf("\nВремя срабатывания будильника"
                    "- %02.2d:%02.2d:%02.2d"
           "\n",
           bcd1bin(&(tmr.hour)),
           bcd1bin(&(tmr.min)),
           bcd1bin(&(tmr.sec)));

// Подключаем свой обработчик прерывания
// будильника, старое значение вектора
// 0x4a сохраняем

     old_4a = _dos_getvect(0x4a);

     _dos_setvect(0x4a, alarm);

// Устанавливаем будильник

     timer(RTC_SET_ALARM, &tmr);

     printf("\nБудильник установлен. Для отмены "
                "и завершения программы нажмите"
                "\nлюбую клавишу...");

     getch();

// Сбрасываем будильник и восстанавливаем
// вектор прерывания будильника

     timer(RTC_CLEAR_ALARM, &tmr);

     _dos_setvect(0x4a, old_4a);

     exit(0);
}

// ---------------------------------
// Преобразование однобайтового
// числа из формата BCD в двоичный
// формат.
// ---------------------------------

int bcd1bin(char *bcd) {

     return( ((*bcd) & 0x0f) +
            10 * (((*bcd) & 0xf0) >> 4) );
}
```

```
// ---------------------------------
// Преобразование двухбайтового
// числа из формата BCD в двоичный
// формат
// ---------------------------------

int bcd2bin(char *bcd) {

     return( bcd1bin(bcd) +
               100 * bcd1bin(bcd + 1) );
}

// ---------------------------------
// Преобразование однобайтового
// числа из двоичного формата
// формат BCD.
// ---------------------------------

int bin1bcd(int bin, char *bcd) {

     int i;

     i = bin / 10;

     *bcd = (i << 4) + (bin - (i * 10));
}

// ---------------------------------
// Программа получает управление
// при срабатывании будильника.
// Ее назначение - выдать звуковой сигнал.
// ---------------------------------

void _interrupt _far alarm(void) {

     BEEP();
     BEEP();
     BEEP();
     BEEP();
     BEEP();
     BEEP();
     BEEP();
}
```

Глава 5

СИСТЕМНЫЙ ТАЙМЕР

Кроме часов реального времени, компьютер (даже простейший IBM PC) содержит устройство, называемое системным таймером. Это устройство подключено к линии запроса на прерывание IRQ0 и вырабатывает прерывание INT 8h приблизительно 18,2 раза в секунду (точное значение - 1193180/65536 раза в секунду).

При инициализации BIOS устанавливает свой обработчик для прерывания таймера. Он каждый раз увеличивает на 1 текущее значение четырехбайтовой переменной, располагающейся в области данных BIOS по адресу 0000:046Ch - счетчик тиков таймера. Если этот счетчик переполняется (прошло более 24 часов с момента запуска таймера), в ячейку 0000:0470h заносится 1.

Стандартный обработчик прерывания таймера осуществляет также контроль за работой двигателей НГМД. Если после последнего обращения к НГМД прошло более 2 секунд, обработчик прерывания выключает двигатель. Ячейка с адресом 0000:0440h содержит время, оставшееся до выключения двигателя. Это время постоянно уменьшается обработчиком прерывания и когда оно становится равно нулю, обработчик выключает двигатель НГМД.

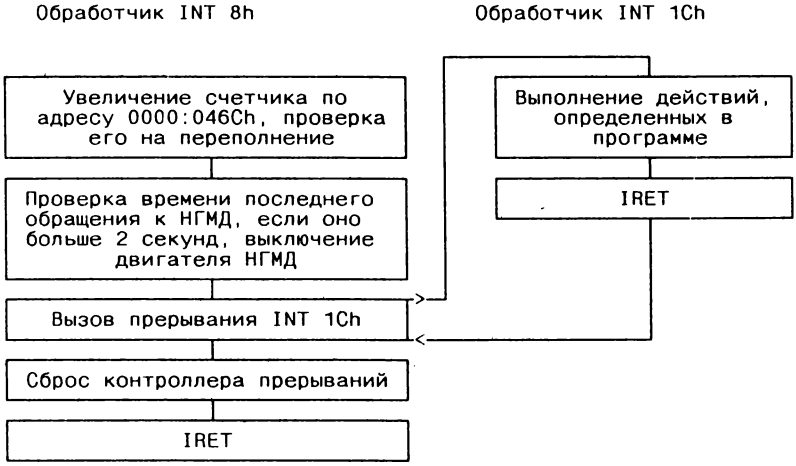
Последнее действие, которое выполняет обработчик прерывания таймера, - вызов прерывания INT 1Ch. После инициализации системы вектор INT 1Ch указывает на команду IRET, т. е. ничего не выполняется. Программа может установить собственный обработчик этого прерывания для того, чтобы выполнять какие-либо периодические действия.

Необходимо отметить, что прерывание INT 1Ch вызывается обработчиком прерывания INT 8h до сброса контроллера прерывания, поэтому во время выполнения прерывания INT 1Ch все аппаратные прерывания запрещены. В частности, запрещены прерывания от клавиатуры.

Обработчик прерывания INT 1Ch должен заканчиваться командой IRET. Если же Вы подготавливаете собственный обработчик для прерывания INT 8h, перед завершением его работы необходимо сбросить контроллер прерываний, например, так:

```
mov  al, 20h
out  20h, al
```

Приведенный ниже рисунок иллюстрирует механизм обработки прерывания таймера:



Таймер обычно реализуется на микросхеме Intel 8253 (для компьютеров IBM PC и IBM XT) или 8254 (для компьютеров IBM AT и IBM PS/2). Следующий раздел книги посвящен описанию микросхемы 8254.

Мы не будем подробно рассказывать о всех возможностях этих микросхем, т. к. обычно используются только несколько режимов работы (а чаще всего один). Полное описание Вы сможете найти в справочной литературе по микросхемам 8253/8254 , а также по их отечественным аналогам К1810ВИ53 и К1810ВИ54.

5.1. Микросхемы таймера 8253/8254

Таймеры 8253 и 8254 состоят из трех независимых каналов, или счетчиков. Каждый канал содержит регистры:

- состояния канала RS (8 разрядов);
- управляющего слова RSW (8 разрядов);

- буферный регистр OL (16 разрядов);
- регистр счетчика CE (16 разрядов);
- регистр констант пересчета CR (16 разрядов).

Каналы таймера подключаются к внешним устройствам при помощи трех линий:

GATE - управляющий вход;
CLOCK - вход тактовой частоты;
OUT - выход таймера.

Регистр счетчика CE работает в режиме вычитания. Его содержимое уменьшается по заднему фронту сигнала CLOCK при условии, что на вход GATE установлен уровень логической 1. В зависимости от режима работы таймера при достижении счетчиком CE нуля тем или иным образом изменяется выходной сигнал OUT.

Буферный регистр OL предназначен для запоминания текущего содержимого регистра счетчика CE без остановки процесса счета. После запоминания буферный регистр доступен программе для чтения.

Регистр констант пересчета CR может загружаться в регистр счетчика, если это требуется в текущем режиме работы таймера.

Как нетрудно догадаться по названию, регистры состояния канала и управляющего слова предназначены соответственно для определения текущего состояния канала и для задания режима работы таймера.

Упрощенная схема взаимодействия регистров канала приведена на рисунке:

Возможны шесть режимов работы таймера. Они разделяются на три типа:

- Режимы 0, 4 - однократное выполнение функций.
- Режимы 1, 5 - работа с перезапуском.
- Режимы 2, 3 - работа с автозагрузкой.

В режиме однократного выполнения функций перед началом счета содержимое регистра констант пересчета CR переписывается в регистр счетчика CE по сигналу CLOCK, если сигнал GATE установлен в 1. В дальнейшем содержимое регистра CE уменьшается по мере прихода импульсов CLOCK. Процесс счета можно приостановить, если подать на вход GATE уровень логического 0. Если затем на вход GATE подать 1, счет будет продолжен дальше. Для повторения выполнения функции необходима новая загрузка регистра CR, т. е. повторное программирование таймера.

При работе с перезапуском не требуется повторного программирования таймера для выполнения той же функции. По фронту сигнала GATE значение константы из регистра CR вновь переписывается в регистр CE, даже если текущая операция не была завершена.

В режиме автозагрузки регистр CR автоматически переписывается в регистр CE после завершения счета. Сигнал на выходе OUT появляется только при наличии на входе GATE уровня логической 1. Этот режим используется для создания программируемых импульсных генераторов и генераторов прямоугольных импульсов (меандра).

В компьютере IBM PC/XT/AT/PS2 задействованы все три канала таймера.

Канал 0 используется в системных часах времени суток (не следует путать с часами реального времени, реализованными на другой микросхеме). Этот канал работает в режиме 3 и используется как генератор импульсов с частотой примерно 18,2 Гц. Именно эти импульсы вызывают аппаратное прерывание INT 8h.

Канал 1 используется для регенерации содержимого динамической памяти компьютера. Выход канала OUT используется для запроса к каналу прямого доступа DMA, который и выполняет обновление содержимого памяти. Вам не следует перепрограммировать этот канал, т. к. это может привести к нарушениям в работе основной оперативной памяти компьютера.

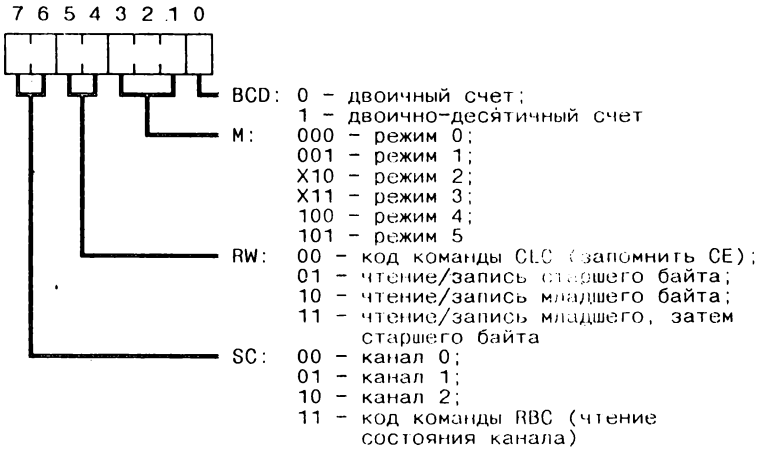
Канал 2 подключен к громкоговорителю компьютера и может быть использован для генерации различных звуков или музыки либо как генератор случайных чисел. Канал использует режим 3 таймера 8253/8254.

5.2. Программирование таймера на уровне портов

Таймеру соответствуют четыре порта ввода/вывода со следующими адресами:

40h - канал 0;
41h - канал 1;
42h - канал 2;
43h - управляющий регистр.

Приведем формат управляющего регистра:



Поле BCD определяет формат константы, использующейся для счета - двоичный или двоично-десятичный. В двоично-десятичном режиме константа задается в диапазоне 1 - 9999. Поле M определяет режимы работы микросхемы 8254:

0 - прерывание от таймера;
1 - программируемый ждущий мультивибратор;
2 - программируемый генератор импульсов;
3 - генератор меандра;
4 - программно-запускаемый одновибратор;
5 - аппаратно-запускаемый одновибратор.

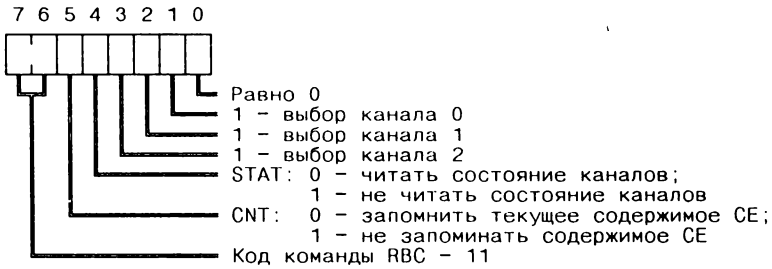
Мы будем рассматривать только режим 3, т. к. именно он используется в каналах 0 и 2.

Поле RW определяет способ загрузки констант через однобайтовый порт. Если в этом поле задано значение 00, это управляющее слово будет использоваться для фиксации текущего содержимого регистров счетчика CE в буферном регистре OL с целью чтения программой.

Код команды CLC - фиксация регистров. Код канала, для которого будет выполняться фиксация, должен быть указан в поле SC. Поля M и BCD при этом не используются.

Поле SC определяет номер канала, для которого предназначено управляющее слово. Если в этом поле задано значение 11, будет выполняться чтение состояния канала.

Формат команды RBC чтения слова состояния канала:

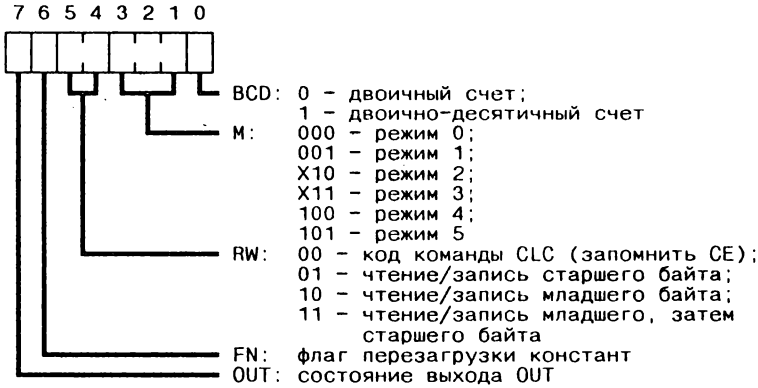


С помощью этой команды Вы можете выполнять операции чтения состояния каналов либо запоминание регистра счетчика CE каналов. Можно выполнять эти операции как для отдельных каналов, так и для всех каналов одновременно, если установить соответствующие биты (1, 2, 3) в 1.

Формат слова состояния канала напоминает формат регистра управляющего слова, исключая два старших разряда - 7 и 6 (см. рисунок на следующей странице).

Разряд FN используется в основном в режимах 1 и 5 для определения, произошла ли загрузка константы из регистра CR в регистр счетчика CE.

Разряд OUT позволяет определить состояние выходной линии канала OUT в момент выполнения команды RBC.



Для программирования канала таймера необходимо выполнить следующую последовательность действий:

- вывести в порт управляющего регистра с адресом 43h управляющее слово;
- требуемое значение счетчика посылается в порт канала (адреса 40h..42h), причем вначале выводится младший, а затем старший байт значения счетчика.

Сразу после этого канал таймера начнет выполнять требуемую функцию.

Для чтения текущего содержимого счетчика CE необходимо выполнить следующее:

- вывести в порт управляющего регистра код команды CLC (команда запоминания содержимого регистра CE);
- вывести в порт управляющего регистра код команды запроса на чтение/запись в регистры канала (поле RW должно содержать 11);
- двумя последовательными командами ввода из порта нужного канала ввести младший и старший байты текущего состояния счетчика CE.

Для чего Вам может понадобиться перепрограммирование каналов таймера?

Если Вам надо повысить точность измерения времени, выполняемого с помощью канала 0 таймера, Вы можете увеличить частоту генерируемых этим каналом импульсов (стандартно 18,2 Гц). По окончании измерений режим работы канала необходимо восстановить для правильного функционирования системы.

Канал 2, подключенный к громкоговорителю, Вы можете использовать для генерации различных звуков или музыки, о чем мы расскажем немного позже. Этот же канал может быть использован для генерации случайных чисел.

Приведем пример программы, отображающей слово состояния и содержимое счетчика для всех трех каналов таймера:

```
#include <stdio.h>
#include <conio.h>

main() {

     unsigned i;

     printf("\n\nКанал 0"
                "\n-------"
                "\n");

// Читаем слово состояния канала,
// команда 0xe2 = 11100010B

     outp(0x43, 0xe2);

     printf("\nСлово состояния канала: %02.2X",
          inp(0x40));

// Читаем текущее состояние регистра счетчика
// канала. Для этого вначале выдаем команду CLC
// для канала 0. Код этой команды - 0x00

     outp(0x43, 0x00);

// Вводим младший и старший байты счетчика
// и отображаем его.

     i = inp(0x40);
     i = (inp(0x40) << 8) + i;

     printf("\nРегистр счетчика:        %04.4X",i);

// Повторяем те же действия для 1 и 2 каналов.

     printf("\n\nКанал 1"
               "\n-------"
                 "\n");
```

```
    outp(0x43, 0xe4);
    printf("\nСлово состояния канала: %02.2X",inp(0x41));

    outp(0x43, 0x40);

    i = inp(0x41);
    i = (inp(0x41) << 8) + i;

    printf("\nРегистр счетчика:       %04.4X",i);

    printf("\n\nКанал 2"
           "\n-------");

    outp(0x43, 0xe8);
    printf("\nСлово состояния канала: %02.2X",inp(0x42));

    outp(0x43, 0x80);

    i = inp(0x42);
    i = (inp(0x42) << 8) + i;

    printf("\nРегистр счетчика:       %04.4X",i);

    exit(0);
}
```

5.3. Средства BIOS для работы с таймером

Для работы с таймером (точнее говоря, для работы с каналом 0 таймера) BIOS содержит две функции прерывания INT 1Ah. Они позволяют прочитать текущее содержимое счетчика и изменить его. Функция 00h предназначена для чтения содержимого счетчика таймера:

На входе:	AH	=	00h.
На выходе:	CX	=	старший байт счетчика;
	DX	=	младший байт счетчика;
	AL	=	0, если с момента перезапуска таймера прошло более 24 часов.

Изменить содержимое счетчика таймера можно с помощью следующей функции:

На входе:	AH	=	01h;
	CX	=	старший байт счетчика;
	DX	=	младший байт счетчика.

На выходе: Не используются.

Функцию чтения таймера можно использовать для организации программной задержки. Так как работа таймера не зависит от производительности процессора, быстродействие системы не будет влиять на формируемую задержку.

Но следует учитывать, что точность формирования задержки определяется частотой обновления счетчика таймера (18,2 Гц) и может оказаться недостаточной для некоторых приложений. Мы подготовили функцию для формирования задержек с помощью таймера:

```
/**
*.Name      tm_delay
*.Title     Формирование задержки по таймеру
*
*.Descr     Эта функция формирует задержку, используя
*           системный таймер.
*
*.Proto     void tm_delay(int ticks)
*
*.Params    int ticks - величина задержки в тиках таймера (за
*                       одну секунду таймер тикает 18,2 раза).
*
*.Return    Ничего
*
*.Sample    tm_samp1.c
**/
#include <dos.h>
#include <conio.h>
void tm_delay(int ticks) {
     _asm {
          push si
          mov  si, ticks
          mov  ah, 0
          int  1ah
          mov  bx, dx
          add  bx, si
delay_loop:
          int  1ah
          cmp  dx, bx
          jne  delay_loop
          pop  si
     }
}
```

Функция использует только одно слово регистра таймера, что позволяет формировать задержки длительностью до 65536 тиков таймера. Приведенная программа демонстрирует использование функции для генерации примерно десятисекундной задержки:

```
#include <stdio.h>
#include "sysp.h"

main() {
     printf("\nДля выполнения программной задержки примерно"
            "\nна 10 секунд нажмите любую клавишу.");
     getch();
     printf("\n\nВремя пошло...");
     tm_delay(18 * 10);
     printf("\nГотово!");
     exit(0);
}
```

BIOS компьютеров IBM AT содержит еще две интересные функции для работы с таймером: 83h и 86h прерывания INT 15h.

Функция 83h позволяет запустить таймер на счет, указав адрес некоторого байта в оперативной памяти. Программа, запустившая таймер, сразу после запуска получает управление. По истечении времени, заданного при запуске таймера, функция устанавливает старший бит указанного байта в единицу, сигнализируя таким образом программе о завершении указанного временного интервала. Программа может также отменить работу таймера в этом режиме.

Эту функцию удобно использовать для организации выполнения каких-либо действий параллельно с отсчетом времени, например, можно ограничить время для ввода пароля.

Приведем формат вызова функции 83h прерывания INT 15h:

На входе: AH = 83h;
AL = код подфункции:
0 - установить интервал, запустить таймер;
1 - отменить работу таймера;
CX = старший байт времени работы счетчика, задается в микросекундах;

DX = младший байт счетчика;
ES:BX = адрес байта, в котором по истечении интервала времени старший бит будет установлен в единицу.

На выходе: Не используются.

Функция 86h специально предназначена для формирования задержек. Она позволяет определять время задержки в микросекундах, что удобно для многих задач. Во время выполнения задержки разрешены прерывания. Формат вызова функции:

На входе: AH = 86h;
CX = старший байт времени задержки, задается в микросекундах;
DX = младший байт времени задержки.

На выходе: Не используются.

5.4. Средства MS-DOS для работы с таймером

MS-DOS использует четыре функции прерывания INT 21h для работы с системным таймером. Эти функции позволяют узнать и установить текущую дату и время. MS-DOS версии 3.30 и более поздних версий при установке времени и даты изменяет также показания часов реального времени.

Для получения текущей даты используется функция 2Ah:

На входе: AH = 2Ah.

На выходе: DL = день (0...31);
DH = ..месяц (1...12);
CX = год (1980...2099);
AL = номер дня недели:
0 - воскресенье;
1 - понедельник;
2 - вторник;
.....
6 - суббота.

Обратите внимание на то, что функция возвращает Вам номер дня недели, который она вычисляет на основе даты.

Для установки даты используйте функцию 2Bh:

На входе:	AH	=	2Bh;
	DL	=	день (0...31);
	DH	=	месяц (1...12);
	CX	=	год (1980...2099).
На выходе:	AL	=	0, если установка выполнена правильно;
	AL	=	FFh, если при установке были заданы неправильные параметры.

Для того чтобы определить текущее время, можно воспользоваться функцией 2Ch:

На входе:	AH	=	2Ch.
На выходе:	CH	=	часы (0...24);
	CL	=	минуты (0...59);
	DH	=	секунды(0...59);
	DL	=	сотые доли секунды (0...99).

Точность времени, полученного при помощи этой функции, определяется таймером (время обновлятся 18,2 раза в секунду).

Для установки времени можно использовать функцию 2Dh:

На входе:	AH	=	2Dh;
	CH	=	часы (0...24);
	CL	=	минуты (0...59);
	DH	=	секунды(0...59);
	DL	=	сотые доли секунды (0...99).
На выходе:	AL	=	0, если установка выполнена правильно;
	AL	=	FFh, если при установке были заданы неправильные параметры.

Стандартные библиотеки трансляторов Microsoft QC 2.5 и C 6.0 содержат многочисленные функции для работы с датой и временем. Они основаны на описанных выше функциях MS-DOS и предоставляют широкие возможности для отображения даты и времени в различных форматах. Подробное описание этих функций и примеры их использования Вы найдете в документации на библиотеки. К сожалению, в этих библиотеках нет функций для организации программных задержек.

5.5. Таймер и музыка

Одно из наиболее распространенных применений таймера - генерация звуковых сигналов и воспроизведение музыки. Таймер позволяет воспроизводить музыку в фоновом режиме, т. е. во время работы программы может звучать музыка.

Как мы уже говорили, канал 2 микросхемы 8254 связан с громкоговорителем компьютера. Однако громкоговоритель не просто соединен с выходом OUT канала 2. Порт вывода 61h также используется для управления громкоговорителем. Младший бит порта 61h подключен ко входу GATE канала 2 таймера. Этот бит при установке в 1 разрешает работу канала, т. е. генерацию импульсов для громкоговорителя.

Дополнительно для управления громкоговорителем используется бит 1 порта 61h. Если этот бит установлен в 1, импульсы от канала 2 таймера смогут проходить на громкоговоритель.

Таким образом, для включения звука надо выполнить следующие действия:

- запрограммировать канал 2 таймера на нужную частоту (т. е. загрузить регистр счетчика канала нужным значением);
- для включения звука установить в 1 два младших бита порта 61h.

Так как остальные 6 битов порта 61h используются для других целей, установка младших битов должна выполняться таким образом, чтобы значения остальных битов не были изменены. Для этого вначале надо считать байт из порта 61h в рабочую ячейку памяти, установить там нужные биты, затем вывести новое значение байта в порт 61h.

Очевидно, что для выключения звука надо сбросить два младших бита порта 61h в 0 (при этом нельзя изменять значение остальных битов этого порта).

Мелодия (одноголосая), как известно, состоит из нот, разделенных или не разделенных паузами. При проигрывании мелодии необходимо для каждой ноты программировать соответствующим образом канал 2 таймера и включать громкоговоритель (с помощью порта 61h) на определенное время, равное длительности

ноты. Затем программа должна выключить динамик и выдержать паузу, если требуется, перед проигрыванием следующей ноты.

Программа может генерировать звуки и другим способом, не используя таймер. Для этого нужно сбросить младший бит порта 61h и, управляя битом 1 этого порта, формировать импульсы для громкоговорителя, т. е. программа должна устанавливать этот бит то в 0, то в 1 с некоторым периодом. Высота генерируемого звука будет соответствовать этому периоду.

Можно также комбинировать эти два способа, получая разнообразные звуковые эффекты.

Для проигрывания мелодии в фоновом режиме можно предложить следующий способ, основанный на использовании периодического прерывания от канала 0 таймера.

Основная идея заключается в использовании прерывания 1Ch, которое вырабатывается таймером с частотой примерно 18,2 Гц. Ваш обработчик этого прерывания осуществляет контроль за выборкой нот из массива, содержащего мелодию, и программирование микросхемы 8254. Например, один раз в полсекунды обработчик проверяет, не пора ли прекратить звучание одной ноты и начать проигрывание следующей ноты. Если пора, он выключает громкоговоритель и перепрограммирует канал 8254 на новую частоту, соответствующую следующей ноте.

Основное преимущество использования таймера для проигрывания мелодии - независимость констант, используемых для программирования канала таймера от производительности системы. Ваша мелодия будет звучать одинаково и на медленной IBM XT, и на Super-AT с процессором 80486, но при условии, что Вы будете использовать таймер и для организации задержек при исполнении мелодии.

Для определения значения, которое должно быть записано в регистр счетчика канала 2 таймера, надо разделить 1193180 на требуемую частоту в герцах.

Мы подготовили функцию, предназначенную для генерации на громкоговорителе тона заданной частоты и длительности:

```
/**
*.Name          tm_sound
*.Title         Формирование тона заданной длительности
*
```

```
*.Descr         Эта функция предназначена для генерации
*               на громкоговорителе тона заданной
*               длительности и частоты.
*
*.Proto         void tm_sound(int freq, int time);
*
*.Params        int freq - частота в герцах;
*               int time - длительность в тиках
*                          таймера (за одну секунду таймер
*                          тикает 18,2 раза).
*
*.Return        Ничего
*
*.Sample        play.c
**/

#include <stdio.h>
#include <conio.h>
#include "sysp.h"

void tm_sound(int freq, int time) {

     int cnt, i;

// Задаем режим канала 2 таймера

     outp(0x43, 0xb6);

// Вычисляем задержку для загрузки в
// регистр счетчика таймера

     cnt = 1193180L / freq;

// Загружаем регистр счетчика таймера - сначала
// младший, затем старший байт

     outp(0x42, cnt & 0x00ff);

     outp(0x42, (cnt &0xff00) >> 8);

// Включаем громкоговоритель. Сигнал от
// канала 2 таймера теперь будет проходить
// на вход громкоговорителя.

     outp(0x61, inp(0x61) | 3);

// Выполняем задержку.

     tm_delay(time);

// Выключаем громкоговоритель.

     outp(0x61, inp(0x61) & 0xfc);
}
```

Если подготовить для этой функции таблицы частот и длительностей, то можно с ее помощью проигрывать простейшие мелодии. В следующем примере мы так и поступили:

```
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

void main(void);
void sound(int, int);

// Массив частот для мелодии

int mary[] = {
     330, 294, 262, 294, 330, 330, 330,
     294, 294, 294, 330, 392, 392,
     330, 294, 262, 294, 330, 330, 330, 330,
     294, 294, 330, 294, 262, 0
};

// Массив длительностей

int del[] = {
     5,   5,   5,   5,   5,   5,   10,
     5,   5,   10,  5,   5,   10,
     5,   5,   5,   5,   5,   5,   5,   5,
     5,   5,   5,   5,   20
};

void main(void) {

  int i;

  for(i=0 ;mary[i] != 0 ;i++)
          tm_sound(mary[i], del[i]);
}
```

Запускайте эту программу и слушайте, как она работает!

Для подготовки таблиц частот по нотам Вам помогут учебники по элементарной теории музыки и следующая таблица, в которой приведены частоты для второй октавы. Для других октав при понижении или повышении тона значения частот надо умножать (при повышении тона) или делить (при понижении тона) на 2.

Нота	*Частота, Гц*
До	261,7
До-диез	277,2
Ре	293,7
Ре-диез	311,1

Ми	329,6
Фа	349,2
Фа-диез	370,0
Соль	392,0
Соль-диез	415,3
Ля	440,0
Ля-диез	466,2
Си	493,9

Приведем еще одну программу, генерирующую звук без использования таймера. Эта программа формирует импульсы при помощи манипуляций с разрядом 1 порта 61h:

```
#include <stdio.h>
#include <conio.h>

#define FREQUENCY 200
#define CYCLES 10000

void main(void);

void main(void) {

     int cnt;

// Во время генерации звука прерывания должны быть запрещены.

     _disable();

     _asm {

// Загружаем количество циклов - периодов
// генерируемых импульсов

          mov   dx, CYCLES

// Отключаем громкоговоритель от таймера

          in    al, 61h
          and   al, 0feh

// Цикл формирования периода

sound_cycle:

// Формируем первый полупериод, подаем
// на громкоговоритель уровень 1

          or    al, 2
          out   61h, al

// Формируем задержку

          mov   cx, FREQUENCY

first: loop  first
```

```
// Формируем второй полупериод, подаем
// на громкоговоритель уровень 0
          and   al,  0fdh
          out   61h, al
// Формируем задержку
          mov   cx, FREQUENCY
second: loop  second
// Если сформированы не все периоды, переходим
// к формированию следующего периода.
          dec   dx
          jnz   sound_cycle
     }
// Разрешаем прерывания.
     _enable();
// Выключаем громкоговоритель.
     outp(0x61, inp(0x61) & 0xfc);
}
```

Так как в этой программе для формирования полупериодов используется задержка с помощью команды LOOP, высота генерируемого тона будет зависеть от производительности системы. Такой зависимости можно избежать, если перед началом работы измерять производительность и соответствующим образом корректировать константу, загружаемую в регистр CX перед вызовом команды LOOP.

Измерение производительности лучше всего выполнять с помощью таймера, определяя время выполнения команды LOOP.

5.6. Генерация случайных чисел

Последнее, что мы сделаем с таймером, - научимся получать от него случайные числа.

Для генерации случайных чисел лучше использовать канал 2 в режиме 3. В регистр счетчика канала мы занесем значение, равное диапазону нужных нам случайных чисел. Например, если мы запишем в регистр число 80 и запустим канал таймера, получаемые случайные числа будут лежать в диапазоне от 0 до 79.

Функция rnd_set() предназначена для начальной инициализации генератора случайных чисел:

```
/**
*.Name          rnd_set
*.Title         Инициализация генератора случайных чисел
*
*.Descr         Эта функция инициализирует канал 2 таймера
*               для использования в качестве генератора
*               случайных чисел.
*
*.Proto         void rnd_set(int bound)
*
*.Params        int bound - верхняя граница для генерируемых
*                           случайных чисел.
*
*.Return        Ничего
*
*.Sample        random.c
**/

#include <stdio.h>
#include <conio.h>

void rnd_set(int bound) {

// Устанавливаем режим 3 для второго канала таймера

     outp(0x43, 0xb6);

// Загружаем регистр счетчика таймера - сначала
// младший, затем старший байт

     outp(0x42, bound & 0x00ff);

     outp(0x42, (bound &0xff00) >> 8);

// Разрешаем работу канала

     outp(0x61, inp(0x61) | 1);
}
```

Через некоторое время после инициализации с помощью функции rnd_get() можно получить готовое случайное число:

```
/**
*.Name          rnd_get
*.Title         Получение от таймера случайного числа
*
*.Descr         Эта функция получает случайное число от
*               таймера, который был предварительно
*               проинициализирован функцией rnd_set.
*
```

```
*.Proto         int rnd_get(void)
*
*.Params        Отсутствуют.
*
*.Return        Случайное число в диапазоне от 0 до
*               уменьшенного на 1 значения, заданного в
*               качестве параметра функции rnd_set().
*
*.Sample        random.c
**/

#include <stdio.h>
#include <conio.h>

int rnd_get(void) {

     int i;

// Выдаем команду CLC для фиксирования
// текущего значения регистра канала 2 таймера

     outp(0x43, 0x86);

// Вводим младший и старший байты счетчика

     i = inp(0x42);
     i = (inp(0x42) << 8) + i;

     return(i);
}
```

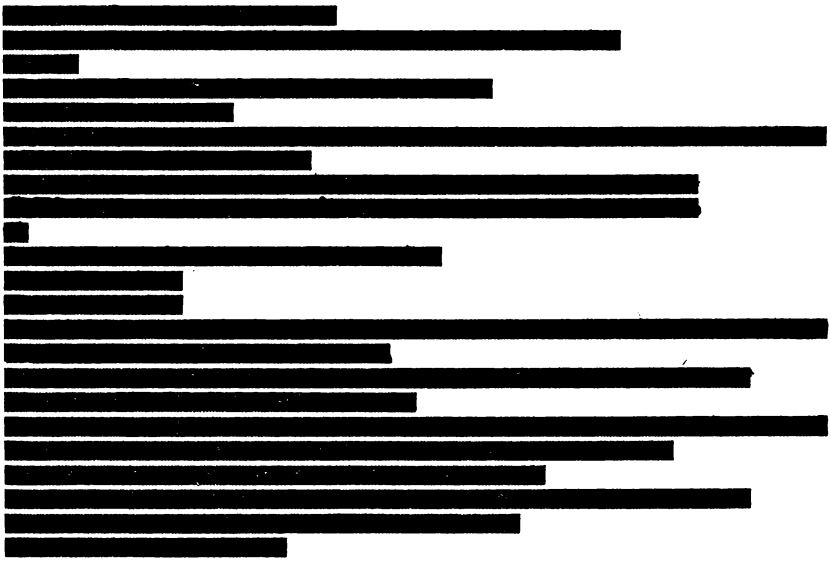
Для иллюстрации использования этих функций мы подготовили следующую программу:

```
#include <stdio.h>
#include "sysp.h"

void main(void);
void main(void) {

     int i, j;

     printf("\nГенератор случайных чисел."
               "\nНажмите любую клавишу,"
               "\nдля завершения работы нажмите CTRL-C."
               "\n");

     for(;;) {

// Устанавливаем диапазон генерации случайных
// чисел и инициализируем таймер

          rnd_set(80);
```

```
// Ожидаем нажатия клавиши.
          getch();
// После нажатия на клавишу получаем
// случайное число
          j = rnd_get();
// Выводим на экран строку символов "",
// длина которой равна полученному случайному
// числу.
          for(i=0; i < j; i++) putchar(219);
          printf("\n");
     }
}
```

Программа получает случайные числа и отображает их в наглядном виде с помощью столбчатой диаграммы:

```
Генератор случайных чисел.
Нажмите любую клавишу,
для завершения работы нажмите CTRL-C.
```



Глава 6

ПОРТ ПОСЛЕДОВАТЕЛЬНОЙ ПЕРЕДАЧИ ДАННЫХ

Глава посвящена порту последовательной передачи данных. Его называют еще портом RS-232-C или асинхронным адаптером RS-232-C. Компьютер IBM PC поддерживает интерфейс RS-232-C не в полной мере, скорее разъем, обозначенный на корпусе компьютера как порт последовательной передачи данных, содержит некоторые из сигналов, входящих в интерфейс RS-232-C и имеющих соответствующие этому стандарту уровни напряжения.

В настоящее время порт последовательной передачи данных используется очень широко. Вот далеко не полный список применений:

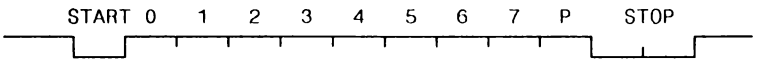
- подключение мыши;
- подключение графопостроителей (плоттеров), сканеров, принтеров, диджитайзеров;
- связь двух компьютеров через порты последовательной передачи данных с использованием специального кабеля и таких программ, как FastWire II или Norton Commander;
- подключение модемов для передачи данных по телефонным линиям;
- подключение к сети персональных компьютеров.

Практически каждый компьютер оборудован хотя бы одним портом для последовательной передачи данных.

6.1. Основные понятия и термины

Последовательная передача данных означает, что данные передаются с использованием единственной линии. При этом биты байта данных передается по очереди с использованием одного провода. Для синхронизации группе битов обычно предшествует специальный стартовый бит, после группы битов следуют бит проверки на четность и один или два стоповых бита. Иногда бит проверки на четность может отсутствовать.

Сказанное иллюстрируется следующим рисунком:



Из рисунка видно, что исходное состояние линии последовательной передачи данных - уровень логической 1. Стартовый бит START сигнализирует о начале передачи данных. Далее передаются биты данных, вначале младшие, затем старшие. Если используется бит четности P, то передается и он. Бит четности имеет такое значение, чтобы в пакете битов общее количество единиц (или нулей) было четно или нечетно.

В самом конце передаются один или два стоповых бита STOP, завершающих передачу байта. Уровень линии передачи снова устанавливается в 1 до прихода следующего стартового бита.

Использование четности, стартовых и стоповых битов определяют протокол передачи данных. Очевидно, что передатчик и приемник данных должны использовать один и тот же протокол, иначе связь будет невозможной.

Другая важная характеристика - скорость передачи данных. Она должна быть одинаковой для передатчика и приемника.

Скорость передачи данных обычно измеряется в бодах. Боды - это количество передаваемых битов в секунду. При этом учитываются и старт/стопные биты, а также бит четности.

Иногда используется другой термин - биты в секунду (bps). Здесь имеется в виду эффективная скорость передачи данных, без учета служебных битов.

6.2. Аппаратная реализация

Компьютер может быть оснащен одним или двумя портами последовательной передачи данных. Эти порты расположены либо на материнской плате, либо на отдельной плате, вставляемой в слоты расширения материнской платы.

Бывают также платы, содержащие 4 или 8 портов последовательной передачи данных. Их часто используют для подключения нескольких компьютеров или терминалов к одному, центральному компьютеру.

В основе последовательного порта передачи данных лежит микросхема Intel 8250. Это универсальный асинхронный приемопередатчик (UART - Universal Asynchronous Receiver Transmitter). Микросхема содержит несколько внутренних регистров, доступных через команды ввода/вывода.

Микросхема 8250 содержит регистры передатчика и приемника данных. При передаче байт записывается в буферный регистр передатчика, откуда затем переписывается в сдвиговый регистр передатчика. Байт "выдвигается" из сдвигового регистра по битам.

Аналогично имеются сдвиговый и буферный регистры приемника.

Программа имеет доступ только к буферным регистрам, копирование информации в сдвиговые регистры и процесс сдвига выполняется микросхемой 8250 автоматически.

Внешние устройства подключаются к порту ввода/вывода через разъем DB25P (имеющий 25 выводов) или DB9P (имеющий 9 выводов). Приведем разводку разъема последовательной передачи данных DB25P:

Номер контакта	*Назначение контакта*	*Вход или выход*
1	Защитное заземление	-
2	Передаваемые данные (Transmitted Data)	Выход
3	Принимаемые данные (Received Data)	Вход
4	Запрос для передачи (Request to send, RTS)	Выход
5	Сброс для передачи (Clear to Send, CTS)	Вход
6	Готовность данных (Data Set Ready, DSR)	-"-
7	Сигнальное заземление	-
8	Детектор принимаемого с линии сигнала (Data Carrier Detect, DCD)	Вход
9 - 19	Не используются	-

20	Готовность выходных данных (Data Terminal Ready, DTR)	Выход
21	Не используется	-
22	Индикатор вызова (Ring Indicator, RI)	Вход
23 - 25	Не используется	-

Наряду с 25-контактным разъемом часто используется 9-контактный разъем:

Номер контакта	*Назначение контакта*	*Вход или выход*
1	Детектор принимаемого с линии сигнала (Data Carrier Detect, DCD)	Вход
2	Принимаемые данные (Received Data)	-"-
3	Передаваемые данные (Transmitted Data)	Выход
4	Готовность выходных данных (Data Terminal Ready, DTR)	-"-
5	Сигнальное заземление	-
6	Готовность данных (Data Set Ready, DSR)	Вход
7	Запрос для передачи (Request to send, RTS)	Выход
8	Сброс для передачи (Clear to Send, CTS)	Вход
9	Индикатор вызова (Ring Indicator, RI)	-"-

Уровни напряжения на линиях разъема составляют для логического нуля -15 вольт, для логической единицы +15 вольт.

Доступ к отдельным линиям возможен через порты ввода/вывода асинхронного адаптера, которые мы рассмотрим в следующем разделе. Там же будет описано назначение отдельных линий разъема.

6.3. Порты асинхронного адаптера

На этапе инициализации системы модуль POST BIOS тестирует имеющиеся асинхронные адаптеры и инициализирует первые два. Их базовые адреса располагаются в области данных BIOS начиная с адреса 0000:0400h.

Первый адаптер COM1 имеет базовый адрес 3F8h и занимает диапазон адресов от 3F8h до 3FFh. Второй адаптер COM2 имеет базовый адрес 2F8h и занимает адреса 2F8h...2FFh.

Асинхронные адаптеры могут вырабатывать прерывания:

COM1 - IRQ4 (соответствует INT 0Ch)
COM2 - IRQ3 (соответствует INT 0Bh)

Рассмотрим назначение отдельных битов этих портов.

Порт 3F8h.

Этот порт соответствует регистру передаваемых данных. Для передачи в порт 3F8h необходимо записать передаваемый байт данных. После приема данных от внешнего устройства они могут быть прочитаны из этого порта.

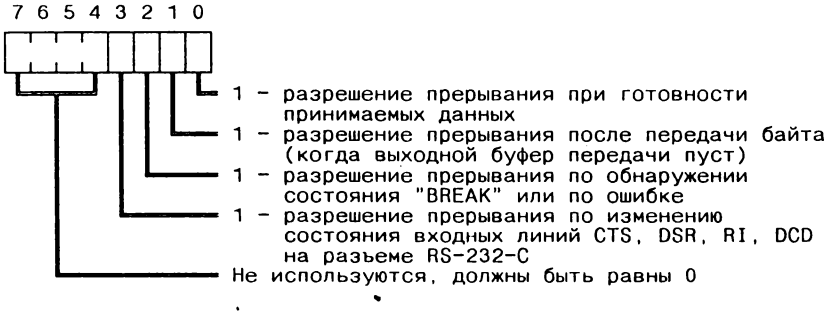
В зависимости от состояния старшего бита управляющего слова, выводимого в управляющий регистр с адресом 3FBh, назначение порта 3F8h может изменяться. Если этот бит равен 0, порт используется для записи передаваемых данных. Если же бит равен 1, порт используется для вывода значения младшего байта делителя частоты тактового генератора. Изменяя содержимое делителя, можно изменять скорость передачи данных. Старший байт делителя записывается в порт 3F9h.

Зависимость скорости передачи данных от значения делителя частоты представлена ниже:

Делитель	Скорость передачи, в бодах	Делитель	Скорость передачи, в бодах
1040	110	24	4800
768	150	12	9600
384	300	6	19200
192	600	3	38400
96	1200	2	57600
48	2400	1	115200

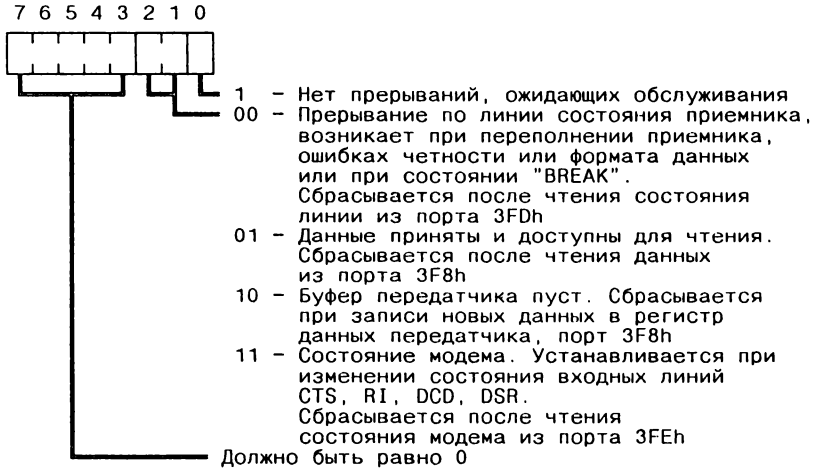
Порт 3F9h

Порт используется как регистр управления прерываниями от асинхронного адаптера или (после вывода в порт 3F9h байта с установленным в 1 старшим битом) для вывода значения старшего байта делителя частоты тактового генератора. В режиме регистра управления прерываниями порт имеет следующий формат:



Порт 3FAh

Регистр идентификации прерывания. По его содержимому программа может определить причину прерывания. Формат регистра:

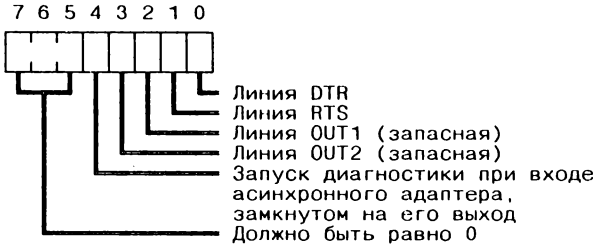


Порт 3FBh

Управляющий регистр, доступен по записи и чтению.

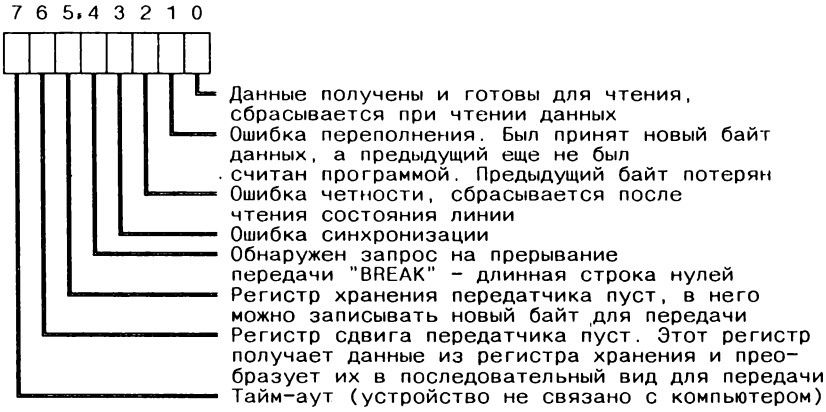
Порт 3FCh

Регистр управления модемом. Управляет состоянием выходных линий DTR, RTS, линий, специфических для модемов OUT1 и OUT2, для запуска диагностики при входе асинхронного адаптера, замкнутом на его выходе. Формат порта:



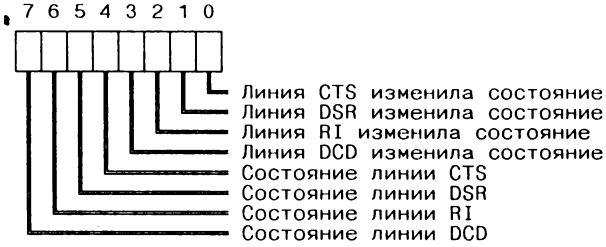
Порт 3FDh

Регистр состояния линии.



Порт 3FEh

Регистр состояния модема.



6.4. Поддержка асинхронного адаптера в BIOS

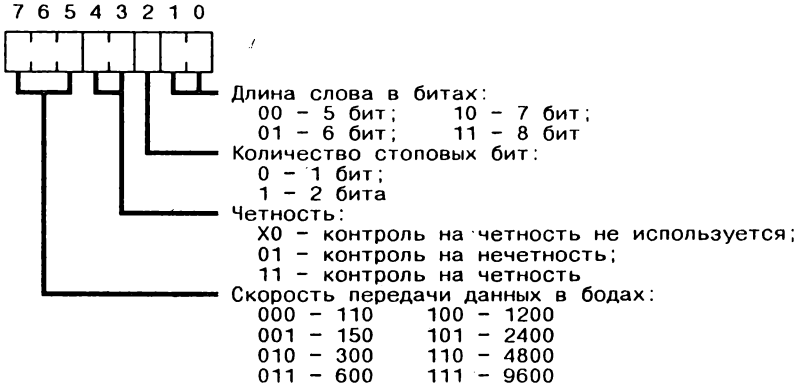
Мы опишем функции BIOS, облегчающие обслуживание двух асинхронных адаптеров, COM1 и COM2. Эти функции доступны через прерывание INT 14h.

Первая функция предназначена для инициализации портов асинхронного адаптера:

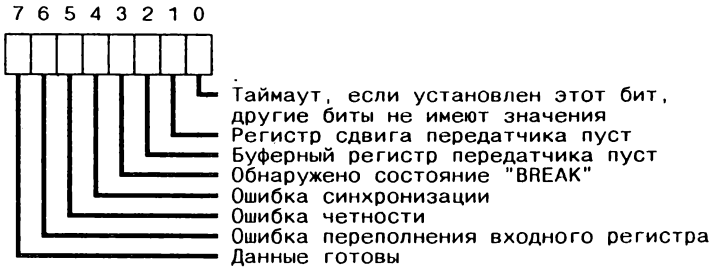
На входе:	AH	=	00h;
	DX	=	номер порта: 0 - COM1, 1 - COM2;
	AL	=	параметры инициализации (см. дальше).

На выходе: AH = состояние порта асинхронного адаптера;
AL = состояние модема.

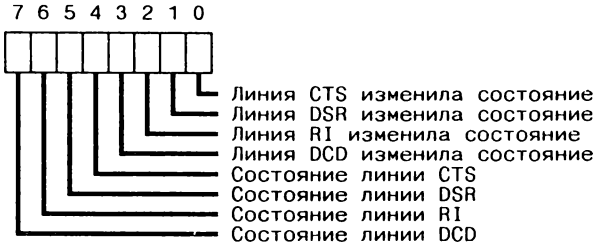
При вызове этой функции регистр AL должен содержать параметры инициализации:



После вызова функции в регистр AH записывается состояние порта асинхронного адаптера:



Регистр AL содержит байт состояния модема:



Для передачи байта используется следующая функция:

На входе:	AH	=	01h;
	DX	=	номер порта: 0 - COM1, 1 - COM2;
	AL	=	передаваемый байт.
На выходе:	AL сохраняется;		
	AH	=	состояние порта асинхронного адаптера, если бит 7 регистра AH установлен в 1, произошла ошибка.

Функция 02h предназначена для приема байта:

На входе:	AH	=	02h;
	DX	=	номер порта: 0 - COM1, 1 - COM2.
На выходе:	AL	=	принятый байт;
	AH	=	состояние порта асинхронного адаптера, если регистр AH не равен 0, произошла ошибка.

Состояние порта асинхронного адаптера можно узнать с помощью функции 03h:

На входе:	AH		03h;
	DX	=	номер порта: 0 - COM1, 1 - COM2.
На выходе:	AH	=	состояние порта асинхронного адаптера;
	AL	=	состояние модема.

6.5. Программирование асинхронного адаптера

К сожалению, MS-DOS не содержит сколько-нибудь серьезной поддержки асинхронного адаптера. Две функции прерывания INT 21h с номерами 3 и 4 предназначены для чтения и записи байтов через асинхронный адаптер. Обе эти функции имеют дело с адаптером COM1 или AUX. Функция 3 получает в регистре AL символ, принятый из адаптера, функция 4 посылает в адаптер символ, записанный в регистр DL.

Основной недостаток функций MS-DOS, предназначенных для работы с адаптером, заключается в отсутствии их функциональной полноты. Используя только функции MS-DOS, Вы не сможете проанализировать ошибочные ситуации и изменить режим работы асинхронного адаптера - нет соответствующих средств.

Функции BIOS, обслуживающие адаптер, более разнообразны. Однако и им присущи недостатки. Например, Вы не сможете установить скорость передачи более 9600 бод или использовать режим фиксации четности. Нет возможности узнать текущий режим асинхронного адаптера, отсутствует поддержка модема.

Поэтому для программирования асинхронного адаптера мы рекомендуем использовать порты ввода/вывода микросхемы 8250.

6.5.1. Инициализация асинхронного адаптера

Первое, что должна сделать программа, работающая с асинхронным адаптером, - установить протокол обмена и скорость передачи данных. После загрузки операционной системы для асинхронных адаптеров устанавливается скорость 2400 бод, не выполняется проверка на четность, используются один стоповый бит и восьмибитовая длина передаваемого символа. Вы можете изменить этот режим командой MS-DOS MODE.

Выполнив ввод из порта 3FBh, программа может получить текущий режим адаптера. Для установки нового режима измените нужные поля и запишите новый байт режима по адресу 3FBh.

Если Вам надо задать новое значение скорости обмена данными, перед записью байта режима установите старший бит этого байта в 1. Затем последовательно двумя командами вывода загрузите делитель частоты тактового генератора. Младший байт запишите в порт 3F8h, старший - в порт 3F9h.

Перед началом работы необходимо также проинициализировать регистр управления прерываниями (порт 3F9h), даже если в Вашей программе не используются прерывания от асинхронного адаптера. Если прерывания Вам не нужны, запишите в этот порт значение нуль.

На этом инициализацию можно считать законченной.

Для того чтобы узнать текущее состояние асинхронного адаптера, Вы можете использовать следующую функцию:

```
/**
*.Name      aux_stat
*.Title     Определение режима асинхронного адаптера
*
*.Descr     Эта функция считывает текущий режим асинхронного
*           порта и записывает его в структуру
```

```
*           с типом AUX_MODE.
*
*.Proto     void aux_stat(AUX_MODE *mode, int port);
*
*.Params    AUX_MODE mode - структура, описывающая
*           протокол и режим работы порта:
*
*      typedef struct _AUX_MODE_ {
*
*       union {
*         struct {
*            unsigned char len : 2, // длина символа
*                 stop         : 1, // число стоп-битов
*                 parity       : 2, // контроль четности
*                 stuck_parity : 1, // фиксация четности
*                 en_break_ctl : 1, // установка перерыва
*                 dlab         : 1; // загрузка регистра
*                                   // делителя
*         } ctl_word;
*         char ctl;
*       } ctl_aux;
*
*       unsigned long baud; // скорость передачи данных
*
*      } AUX_MODE;
*
*           int port - номер асинхронного адаптера:
*              0 - COM1, 1 - COM2
*
*.Return    Ничего
*
*.Sample    aux_test.c
**/
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

void aux_stat(AUX_MODE *mode, int port) {

     unsigned long b;

// Запоминаем режим адаптера

     mode->ctl_aux.ctl = (char)inp(0x3fb - 0x100 * port);

// Устанавливаем старший бит режима
// для считывания текушей скорости передачи

     outp(0x3fb - 0x100 * port, mode->ctl_aux.ctl | 0x80);

// Считываем значение регистра делителя

     b = inp(0x3f9 - 0x100 * port); b = b << 8;
     b += inp(0x3f8 - 0x100 * port);
```

```
// Преобразуем его в боды
     switch (b) {
          case 1040: b = 110; break;
          case 768: b = 150; break;
          case 384: b = 300; break;
          case 192: b = 600; break;
          case 96: b = 1200; break;
          case 48: b = 2400; break;
          case 24: b = 4800; break;
          case 12: b = 9600; break;
          case 6: b = 19200; break;
          case 3: b = 38400; break;
          case 2: b = 57600; break;
          case 1: b = 115200; break;
          default: b=0; break;
     }
     mode->baud = b;
// Восстанавливаем состояние адаптера
     outp(0x3fb - 0x100 * port, mode->ctl_aux.ctl & 0x7f);
}
```

Прочитав состояние адаптера, Вы можете изменить нужные Вам поля в структуре AUX_MODE и вызвать функцию aux_init() для изменения параметров адаптера:

```
/**
*.Name       aux_init
*.Title      Инициализация асинхронного адаптера
*
*.Descr      Эта функция инициализирует асинхронные адаптеры,
*            задавая протокол обмена данными и скорость
*            обмена данными.
*
*.Proto      int aux_init(AUX_MODE *mode, int port, int imask);
*
*.Params     AUX_MODE *mode - указатель на структуру,
*                  описывающую протокол и режим работы порта;
*
*            int port - номер асинхронного адаптера:
*               0 - COM1, 1 - COM2
*
*            int imask - значение для регистра маски прерываний
*
*.Return     0 - инициализация выполнена успешно;
*            1 - ошибки в параметрах инициализации.
*
*.Sample     aux_test.c
**/
```

```
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

int aux_init(AUX_MODE *mode, int port, int imask) {

     unsigned div;
     char ctl;

// Вычисляем значение для делителя

     switch (mode->baud) {
          case 110: div = 1040; break;
          case 150: div = 768; break;
          case 300: div = 384; break;
          case 600: div = 192; break;
          case 1200: div = 96; break;
          case 2400: div = 48; break;
          case 4800: div = 24; break;
          case 9600: div = 12; break;
          case 19200: div = 6; break;
          case 38400: div = 3; break;
          case 57600: div = 2; break;
          case 115200: div =1; break;
          default: return(-1); break;
     }

// Записываем значение делителя частоты

     ctl = inp(0x3fb - 0x100 * port);
     outp(0x3fb - 0x100 * port, ctl | 0x80);

     outp(0x3f9 - 0x100 * port, (div >> 8) & 0x00ff);
     outp(0x3f8 - 0x100 * port, div & 0x00ff);

// Записываем новое управляющее слово

     outp(0x3fb - 0x100 * port, mode->ctl_aux.ctl & 0x7f);

// Устанавливаем регистр управления прерыванием

     outp(0x3f9 - 0x100 * port, imask);

     return(0);
}
```

6.5.2. Передача данных

Перед записью байта данных в регистр передатчика нужно убедиться, что регистр хранения передатчика свободен, т. е. убедиться в том, что передача предыдущего символа завершена. Признаком свободы регистра передатчика является установленный в 1 бит 5 регистра состояния линии с адресом 3FDh.

Следующая функция ждет окончания передачи текущего символа, затем посылает в асинхронный адаптер следующий символ:

```
/**
*.Name       aux_outp
*.Title      Вывод символа в асинхронный адаптер
*
*.Descr      Эта функция дожидается готовности
*            передатчика и посылает символ.
*
*.Proto      void aux_outp(char chr, int port);
*
*.Params     char chr - посылаемый символ;
*
*            int port - номер асинхронного адаптера:
*               0 - COM1, 1 - COM2
*
*.Return     Ничего
*
*.Sample     aux_test.c
**/
#include <stdio.h>
#include <conio.h>
#include "sysp.h"

void aux_outp(char chr, int port) {

     unsigned status_reg, out_reg;

     status_reg = 0x3fd - 0x100 * port;
     out_reg = status_reg - 5;

     while( (inp(status_reg) & 0x20) == 0 );

     outp(out_reg, chr);
}
```

6.5.3. Прием данных

Аналогично передаче данных перед вводом символа из порта приемника 3F8h следует убедиться, что бит 0 порта 3FDh установлен в 1, т. е. что символ принят из линии и находится в буферном регистре приемника. Для приема данных мы подготовили следующую функцию:

```
/**
*.Name       aux_inp
*.Title      Ввод символа из асинхронного адаптера
*
*.Descr      Эта функция дожидается готовности приемника
```

```
*           и вводит символ из асинхронного адаптера.
*
*.Proto     char aux_inp(int port);
*
*.Params    int port - номер асинхронного адаптера:
*              0 - COM1, 1 - COM2
*
*.Return    Принятый символ
*
*.Sample    aux_test.c
**/

#include <stdio.h>
#include <conio.h>
#include "sysp.h"

char aux_inp(int port) {

     unsigned status_reg, inp_reg;

     status_reg = 0x3fd - 0x100 * port;
     inp_reg = status_reg - 5;
     while( (inp(status_reg) & 1) == 0 );

     return(inp(inp_reg));
}
```

6.5.4. Пример программы передачи данных

Приведем пример программы, использующей описанные выше функции для изменения скорости передачи данных и для проверки асинхронного адаптера. Для правильной работы программы выход асинхронного адаптера должен быть соединен с его входом.

```
// Программа работает с асинхронным адаптером COM1.
// Для правильной работы необходимо замкнуть
// вместе контакты 2 и 3 разъема COM1.

#include <stdio.h>
#include <conio.h>
#include "sysp.h"

void main(void);
void main(void) {

     AUX_MODE amd;

     aux_stat(&amd, 0);
     printf("\nСостояние порта COM1:"
                "\nКод длины символа:    %d"
                "\nКод числа стоп-битов: %d"
                "\nКонтроль четности:    %d"
                "\nСкорость передачи:    %lu",
```

```
                amd.ctl_aux.ctl_word.len,
                amd.ctl_aux.ctl_word.stop,
                amd.ctl_aux.ctl_word.parity,
                (unsigned long)amd.baud);

    amd.baud = 115200;

    aux_init(&amd, 0, 0);

    aux_stat(&amd, 0);
    printf("\nСостояние порта COM1:"
                "\nКод длины символа:    %d"
                "\nКод числа стоп-битов: %d"
                "\nКонтроль четности:    %d"
                "\nСкорость передачи:    %lu",
                amd.ctl_aux.ctl_word.len,
                amd.ctl_aux.ctl_word.stop,
                amd.ctl_aux.ctl_word.parity,
                (unsigned long)amd.baud);

    printf("\n\nТестирование асинхронного адаптера."
                "\nНажимайте клавиши!"
                "\nДля завершения работы нажмите CTRL-C"
                "\n");

    for(;;) {

// Вводим символ с клавиатуры и передаем его
// в асинхронный адаптер

          aux_outp(getch(), 0);

// Вводим символ из асинхронного адаптера и
// отображаем его на экране

          putchar(aux_inp(0));
    }
}
```

6.5.5. Использование прерываний

Так как процесс последовательной передачи данных протекает достаточно медленно, имеет смысл выполнять его в фоновом режиме, используя прерывания по окончании передачи или приема символа. Напомним, что порту COM1 соответствует аппаратное прерывание INT 0Ch, а COM2 - INT 0Bh.

Для разрешения прерываний необходимо установить в 1 биты порта управления прерываниями 3F9h, соответствующие тем прерываниям, которые мы желаем обрабатывать.

Когда произошло прерывание, программа-обработчик прерывания должна проанализировать причину прерывания, прочитав содержимое порта идентификации прерывания с адресом 3FAh.

Не забудьте, что в конце обработчика аппаратного прерывания должна находится последовательность команд:

```
mov al, 20h
out 20h, al

iret
```

Может случиться так, что одновременно произойдет несколько прерываний. В этом случае бит 0 регистра идентификации прерывания будет установлен в 1. Если такая ситуация имеет место, перед завершением обработки прерывания Вам надо снова прочитать регистр идентификации прерывания и обработать следующее прерывание. Так следует поступать до тех пор, пока бит 0 регистра идентификации прерывания не станет равным нулю.

Глава 7

ПРИНТЕР

В этой главе мы расскажем об использовании матричных принтеров и способах их подключения к компьютеру.

Матричные принтеры распространены наиболее широко, в основном из-за их низкой стоимости. Если у Вас нет потребности в печати большого объема документации (тысячи листов) или Вам не нужно типографское качество получаемых документов, используйте матричные принтеры.

Существуют две группы матричных принтеров, различающиеся по системе используемых команд, - это принтеры, совместимые с принтерами Epson, и принтеры, совместимые с IBM Proprinter. Принтеры некоторых третьих фирм - производителей компьютерного оборудования (не Epson и не IBM) обычно выполняют эмуляцию команд обеих или одной из этих групп в зависимости от установки переключателей конфигурации.

Кроме того, различные модели принтеров, например производства фирмы Epson, также отличаются набором команд. Как правило, обеспечивается совместимость по командам снизу вверх, т. е. более новые модели поддерживают команды предыдущих моделей принтеров.

Очень распространены принтеры серии Epson FX: FX-80, FX-850, FX-1050. Печатающие головки этих принтеров имеют девять иголок, поэтому качество печати принтеров серии FX оставляет желать лучшего. Принтеры серии Epson LQ используют для печати 24 иголки, кроме того, некоторые модели способны печатать цветные изображения (например, Epson LQ-2550). Качество печати принтеров LQ сравнимо с типографским.

7.1. Подключение принтера к компьютеру

Принтер подключается к компьютеру двумя способами: либо через асинхронный адаптер, рассмотренный ранее, либо через порт параллельной передачи данных. Возможно подключение к одному компьютеру сразу нескольких принтеров, причем принте-

ры могут быть подключены одновременно и к параллельному порту, и к асинхронному последовательному адаптеру.

Для подключения принтера к последовательному порту компьютера принтер должен быть оборудован специальным последовательным интерфейсом. Кроме того, необходимо использовать специальный кабель.

Если Вы подсоедините принтер к последовательному порту при помощи кабеля, предназначенного для работы с параллельным портом, то это может привести к повреждениям в оборудовании компьютера или принтера. Внимательно читайте раздел документации на принтер, посвященный подключению его к компьютеру.

В этой главе мы сделаем упор на использование порта параллельной передачи данных, т. к. именно этот вариант подключения принтера наиболее распространен.

7.2. Работа параллельного принтерного порта

BIOS может работать с тремя параллельными принтерными портами. В процессе тестирования и инициализации системы BIOS находит работоспособные принтерные порты и записывает их базовые адреса в таблицу. Таблица адресов располагается в области данных BIOS по адресу 0000:0408h. Возможны следующие значения базовых адресов:

378h - принтерный порт LPT1;
278h - принтерный порт LPT2;
3BCh - принтерный порт на плате адаптера монохромного дисплея.

Принтерные порты могут вырабатывать запросы на прерывание:

LPT1 - IRQ7, INT 0Fh;
LPT2 - IRQ5, INT 0Dh.

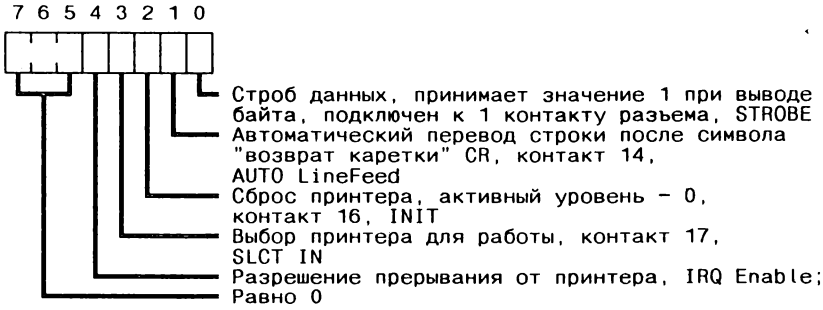
Каждый принтерный порт (принтерный адаптер) обслуживают несколько портов ввода/вывода. Рассмотрим их назначение.

Порт 378h

Этот порт предназначен для записи выводимого на принтер байта данных. Возможно также чтение только что записанного байта.

Порт 37Ah

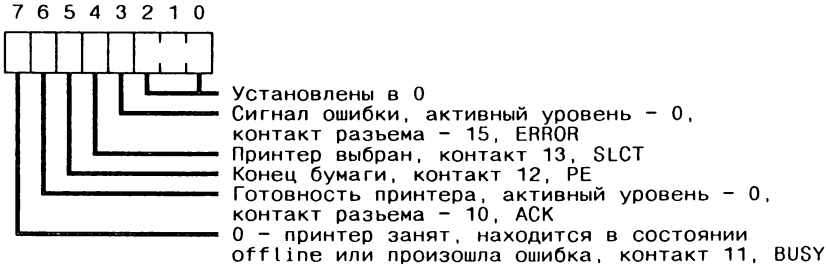
Порт управления принтером, доступен для чтения и записи:



Если прерывания от принтера разрешены, они вырабатываются, когда сигнал готовности принтера ACK (контакт разъема 10) принимает уровень логического нуля

Порт 379h

Порт состояния принтера, доступен только для чтения:



Обычно редко приходится работать с принтером на уровне портов ввода/вывода, т. к. достаточно использовать функции BIOS или MS-DOS, предназначенные для этого. Приведенная выше информация может пригодиться Вам для разработки собственного драйвера принтера или для подключения к принтерному порту

какого-либо другого устройства ввода/вывода, например аналого-цифрового преобразователя.

Для тех, кто будет использовать принтерный порт для подключения аппаратуры, приведем таблицу назначения контактов разъемов принтерного порта (контакт PC) на компьютере и контактов разъема непосредственно на принтере (контакт принтера):

Контакт PC	*Контакт принтера*	*Назначение*	*Вход/выход*
1	1	Строб (STROBE)	Выход, инверсия
2	2	Данные, бит 0	Выход
3	3	-"- бит 1	-
4	4	-"- бит 2	-
5	5	-"- бит 3	-
6	6	-"- бит 4	-
7	7	-"- бит 5	-
8	8	-"- бит 6	-
9	9	-"- бит 7	-
10	10	Подтверждение, ACK	Вход, инверсия
11	11	Занятость, BUSY	Вход
12	12	Конец бумаги, PE	-
13	13	Выбор, SLCT	-
14	14	Авт. перевод строки, Auto Line Feed	Выход, инверсия
15	32	Ошибка, ERROR	Вход, инверсия
16	31	Сброс принтера, INIT	Выход, инверсия
17	36	Принтер выбран, SLCT IN	-
18-25	16,17, 19-30,33	Земля	-

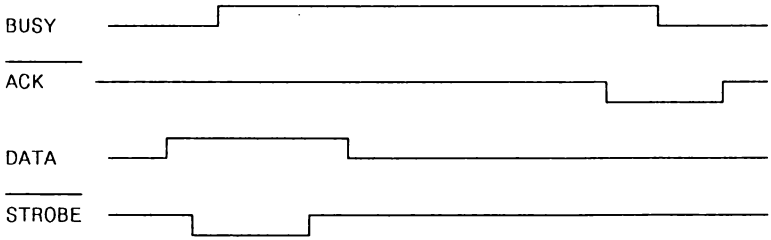
Для сигналов, отмеченных в таблице словом "инверсия", используется уровень логического нуля в активном состоянии сигнала.

Используя принтерный порт для управления внешними устройствами, будьте осторожны и выполняйте все правила заземле-

ния устройств. Если Ваше устройство не заземлено или заземлено неправильно, принтерный порт может выйти из строя.

Следует также учитывать, что нагрузка на выходную линию принтерного порта не должна превышать одного входа TTL.

Если по каким-либо причинам Вы пожелаете работать с принтером через порты ввода/вывода, Вам необходимо изучить временную диаграмму принтерного порта. Она представлена на следующем рисунке:



Для того чтобы вывести символ на принтер, программа вначале должна убедится, что уровень сигнала на линии BUSY (бит 7 порта 379h) равен нулю, а уровень сигнала на линии ACK (бит 6 порта 379h) - единице. После этого следует установить код выводимого символа на линиях DATA (порт 378h).

Затем не ранее чем через 0,5 мкс линию STROBE (бит 0 порта 37Ah) необходимо перевести в состояние логического нуля. При этом выводимый символ запишется во внутренний буфер принтера. Уровень логического нуля необходимо удерживать в течение как минимум 0,5 мкс. Это время нужно для того, чтобы символ записался в буфер принтера. После истечения интервала времени линию STROBE нужно опять перевести в состояние логической единицы.

После того как программа установит линию STROBE в состояние логического нуля, выходная линия принтера BUSY устанавливается в единицу, сигнализируя о том, что принтер занят обработкой полученного символа и временно не может принимать другие символы.

Когда принтер полность обработает выведенный символ, линия ACK перейдет в состояние нуля. Приблизительно через 5 мкс после этого линия BUSY также перейдет в состояние нуля.

Еще через 5 мкс линия ACK примет состояние единицы. Теперь принтер готов принят следующий символ распечатываемых данных.

7.3. Средства BIOS для работы с принтером

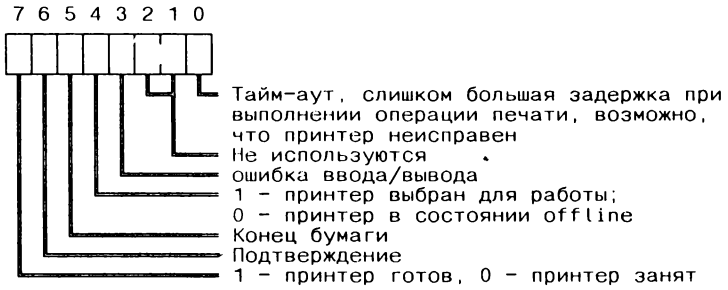
BIOS использует для работы с принтером функции 0, 1, 2 прерывания INT 17h.

Функция 00h предназначена для печати одного символа:

На входе:	AH	=	00h;
	AL	=	ASCII-код символа для печати;
	DX	=	номер принтера: 0, 1 или 2.
На выходе:	AH	=	слово состояния принтера (см. ниже).

Эта функция выводит на принтер один символ, заданный в регистре AL. В регистр DX необходимо записать номер используемого принтера, для LPT1 это 0, для LPT2 - 1 и т. д.

После выполнения прерывания регистр AH будет содержать слово состояния, имеющее следующий формат:



Вызвав функцию 0 прерывания INT 17h, программа должна проверить отдельные биты слова состояния и убедиться в том, что вывод байта произошел без ошибок. Наиболее часто оператор забывает перевести принтер в состояние ONLINE, либо вставить бумагу, либо вообще включить принтер. В этом случае целесообразно напомнить оператору о необходимости выполнения этих действий и затем повторить печать символа.

Если принтер неисправен, программа должна предоставить оператору возможность отменить печать текста. Ниже приведен

пример программы, выполняющей печать текста и анализирующей ошибки, которые могут возникнуть в процессе печати.

Обратите внимание на бит 1 байта состояния - тайм-аут. Если принтер находится в состоянии OFFLINE, функция 0 прерывания INT 17h ожидает некоторое время готовности принтера, после чего, если принтер так и не перешел в состояние готовности, устанавливает бит 1 в байте состояния. Область данных BIOS по адресу 0000h:0478h содержит четыре байта, которые используются в качестве счетчиков времени при ожидании готовности принтера.

Прерывание INT 17h имеет еще две функции, выполняющие инициализацию принтера и получающие текущее состояние принтера.

Функция 01h инициализирует принтер:

На входе:	AH	=	01h;
	DX	=	номер принтера: 0, 1 или 2.
На выходе:	AH	=	слово состояния принтера.

Эта функция выполняет аппаратный сброс принтера. Если Вы загрузили в принтер какой-либо шрифт (например, кириллицу), после сброса загрузку шрифта придется выполнять заново. Поэтому не следует выполнять сброс принтера, если это действительно не требуется. Обычно принтер приходится сбрасывать либо перед настройкой его на заданный режим работы, которая выполняется один раз, либо при изменении этого режима.

Слово состояния принтера может быть получено с помощью функции 02h:

На входе:	AH	=	02h;
	DX	=	номер принтера: 0, 1 или 2.
На выходе:	AH	=	слово состояния принтера.

Эту функцию удобно использовать перед началом печати для определения готовности принтера к работе.

Приведем программу, которая распечатывает содержимое файла с использованием функции 0 прерывания INT 17h.

Программа считывает содержимое файла, открытого в двоичном режиме по байтам. Считанные файлы передаются в качестве параметра функции printchar(), которая и выводит их на принтер. После вызова прерывания INT 17h проверяется состояние

принтера и в случае, когда произошла ошибка ввода/вывода, вызывается обработчик ошибки - функция error(). Эта функция выводит на экран состояние принтера (в развернутом виде, с объяснением каждого бита байта состояния) и запрашивает оператора о дальнейших действиях.

Если оператор может устранить причину ошибки (перевести принтер в состояние ONLINE, вставить бумагу, если она кончилась и т. д.), он нажимает любую клавишу, кроме ESC, и тогда функция error() возвращает 0. В противном случае возвращается значение 1.

Если оператор решил повторить печать, и соответственно если функция error() возвратила значение 0, функция printchar() повторяет печать символа. В противном случае выдается сообщение об ошибке и работа программы завершается.

Итак, приведем исходный текст программы печати содержимого текстовых файлов:

```
#include <dos.h>
#include <stdio.h>

union REGS rg;

int main(int argc, char *argv[]) {

     FILE *srcfile;

// Открываем файл, заданный первым параметром
// в командной строке.
// Если при запуске программы оператор забыл
// указать имя файла, выводим напоминающее сообщение.

     if( (srcfile = fopen( argv[1], "rb" )) == NULL ) {
          printf("\nЗадайте имя файла в качестве параметра");
          exit(-1);
     }

// Читаем файл по одному символу, полученный из файла
// символ выводим на принтер при помощи функции printchar().

     for(;;) {
          printchar(fgetc(srcfile));
          if(feof(srcfile)) break;
     }

// Закрываем файл.

     fclose(srcfile);
}
```

```
// ------------------------------------
// Эта функция выводит один символ
// на первый принтер (LPT1)
// ------------------------------------

int printchar(int chr) {

     int status;

// Повторяем в цикле выдачу символа на принтер
// до тех пор, пока он не будет выведен без
// ошибок, либо пока оператор не отменит
// распечатку файла.

     for(;;) {

// Дублируем распечатываемый символ на экране

          putch(chr);

// Вызываем функцию 0 прерывания INT 17h -
// распечатка символа на принтере.
// В регистре DX задаем номер принтера LPT1 - это 0.

          rg.h.ah = 0;
          rg.h.al = chr;
          rg.x.dx = 0;

          int86(0x17, &rg, &rg);

// Запоминаем байт состояния принтера
// после вывода символа.

          status = rg.h.ah;

// Проверяем наличие ошибок. Нас интересуют биты:
//    0 - таймаут (задержка при печати слишком велика)
//    3 - ошибка ввода/вывода
//    4 - принтер в состоянии ONLINE (1) или OFFLINE (0)
//    5 - конец бумаги

          if((status & 0x39) != 0x10) {

// Вызываем функцию обработки ошибки error(). Эта
// функция возвращает 0, если оператор желает
// повторить печать символа, или 1 - если
// оператор отменяет печать.

               if(error(chr, status)) {
                    printf("\nПечать завершилась аварийно");
                    exit(-2);
               }
          }
          else break;
     }
}
```

```
// ------------------------------------
// Функция выводит на экран состояние
// принтера и запрашивает у оператора
// требуемые действия - повторить
// печать символа или отменить печать.
// ------------------------------------

int error(char chr, int status) {

// Выводим состояние принтера после ошибки

          printf("\n0шибка принтера %02.2X"
          "\n\nСостояние принтера:"
            "\n--------------------", status);

          if(status & 1)
               printf("\nТайм-аут при печати");

          if(status & 8)
               printf("\n0шибка ввода/вывода");

          if(!(status & 0x10))
               printf("\nПринтер находится в состоянии
                      "OFFLINE");

          if(status & 0x20)
               printf("\nКонец бумаги");

          printf("\n\nДля отмены печати нажмите клавишу ESC,"
                    "\nдля повтора - любую другую клавишу\n");

          if(getch() == 27) return(1);
          else return(0);
}
```

7.4. Средства MS-DOS для работы с принтером

Для печати символа на стандартном печатающем устройстве LPT1 (он же PRN) Вы можете использовать функцию 05h прерывания MS-DOS INT 21h:

На входе:	AH	=	05h;
	DL	=	ASCII-код символа для печати.
На выходе:	AH	=	слово состояния принтера (см. ниже).

Команда MS-DOS MODE может переназначить стандартное устройство печати LPT1 на асинхронный последовательный порт:

```
MODE LPT1:=COM1
```

Мы подготовили еще одну программу распечатки содержимого файла, но уже при помощи прерывания MS-DOS:

```
#include <dos.h>
#include <stdio.h>

union REGS rg;

int main(int argc, char *argv[]) {
     FILE *srcfile;

// Открываем файл, заданный первым параметром в командной
// строке. Если при запуске программы оператор забыл
// указать имя файла, выводим напоминающее сообщение.

     if( (srcfile = fopen( argv[1], "rb" )) == NULL ) {
          printf("\nЗадайте имя файла в качестве параметра");
          exit(-1);
     }

// Читаем файл по одному символу, полученный из файла
// символ выводим на принтер при помощи функции printchar().

     for(;;) {
          printchar(fgetc(srcfile));
          if(feof(srcfile)) break;
     }

// Закрываем файл.

     fclose(srcfile);
}

// ------------------------------------
// Эта функция выводит один символ
// на стандартный принтер (LPT1)
// ------------------------------------

int printchar(int chr) {

// Дублируем распечатываемый символ на экране

          putch(chr);

// Вызываем функцию 5 прерывания INT 21h -
// распечатка символа на принтере.

          rg.h.ah = 5;
          rg.h.dl = chr;

          int86(0x21, &rg, &rg);
}
```

Заметьте, что функция 05h прерывания INT 21h не возвращает состояния принтера при ошибке ввода/вывода. Вместо этого вызывается стандартный обработчик критических ошибок MS-DOS, который выводит на экран знакомое Вам сообщение:

```
Write fault error writing device PRN
Abort, Retry, Ignore, Fail?
```

Вы можете ответить Retry, нажав клавишу "R", и MS-DOS попытается повторить печать символа. Если ответить Abort (нажав клавишу "A"), MS-DOS завершит работу Вашей программы.

Поэтому приведенная выше программа не содержит обработчика ошибочных ситуаций error(). Если Вас не устраивают действия, выполняемые стандартным обработчиком критических ошибок MS-DOS, Вы можете составить собственный. В третьей книге первого тома "Библиотеки системного программиста" мы рассказывали Вам о создании и подключении собственного обработчика критических ошибок.

Более интересные возможности по управлению процессом печати предоставляет программа резидентного спулера печати PRINT.COM. Вы знаете, что команда PRINT предназначена для выполнения печати в фоновом режиме. Оказывается, что, если запущена программа PRINT, другие программы могут взаимодействовать с ней, управляя процессом печати.

Для связи со спулером печати можно использовать несколько функций прерывания INT 2Fh:

На входе:	AH	=	01h;
	AL	=	номер выполняемой операции.
На выходе:	AH	=	00 - спулер печати не установлен, но его можно установить, запустив программу PRINT;
			01 - спулер печати не установлен и его установка невозможна (система не содержит ни одного принтера);
			FFh - спулер установлен.

Приведем форматы регистров для выполнения различных операций со спулером печати.

На входе:	AH	=	01h;
	AL	=	0 - проверить установку спулера печати.
На выходе:	AH	=	00 - спулер печати не установлен, но его можно установить, запустив программу PRINT;

01 - спулер печати не установлен и
его установка невозможна;
FFh - спулер установлен.

На входе: AH = 01h;
AL = 1 - передача файла спулеру для печати;
DS:DX = адрес управляющего блока:

Смещение	Длина	
(+0)	1	уровень запроса, равен 0;
(+1)	4	FAR-адрес строки в формате ASCIIZ, содержащей путь файла.

На выходе: AH = 00 - спулер печати не установлен,
но его можно установить, запустив
программу PRINT;
01 - спулер печати не установлен
и его установка невозможна;
FFh - спулер установлен.

На входе: AH = 01h;
AL = 2 - отменить печать файла;
DS:DX = адрес строки в формате ASCIIZ,
содержащей имя файла, удаляемого
из очереди для печати.

На выходе: AH = 00 - спулер печати не установлен,
но его можно установить, запустив
программу PRINT;
01 - спулер печати не установлен
и его установка невозможна;
FFh - спулер установлен.

На входе: AH = 01h;
AL = 3 - отменить печать всех файлов.

На выходе: AH = 00 - спулер печати не установлен,
но его можно установить, запустив
программу PRINT;
01 - спулер печати не установлен
и его установка невозможна;
FFh - спулер установлен.

На входе: AH = 01h;

	AL	=	4 - определить состояние спулера и заблокировать спулер.
На выходе:	DS:SI	=	адрес очереди печати (массив строк в формате ASCIIZ, конец массива отмечен строкой, состоящей из 0;
	DX	=	количество ошибок при попытке напечатать последний символ;
	AH	=	00 - спулер печати не установлен, но его можно установить, запустив программу PRINT; 01 - спулер печати не установлен и его установка невозможна; FFh - спулер установлен.
На входе:	AH	=	01h;
	AL	=	5 - разблокировать спулер для продолжения печати.
На выходе:	AH	=	00 - спулер печати не установлен, но его можно установить, запустив программу PRINT; 01 - спулер печати не установлен и его установка невозможна; FFh - спулер установлен.

Если после вызова перечисленных выше функций флаг переноса CF установлен в 1, регистр AX содержит код ошибки:

1	Неправильный код функции
2	Файл не найден
3	Путь не найден
4	Слишком много открытых файлов
5	Доступ запрещен
6	Неправильный индекс (handle)
8	Переполнение очереди
9	Занято
0Ch	Слишком длинный путь и имя файла (больше 64 байтов)
0Fh	Неправильное определение диска

7.5. Установка переключателей конфигурации

Как правило, матричные принтеры позволяют устанавливать режим своей работы с помощью переключателей режима. Для доступа к этим переключателям Вам не надо разбирать принтер или снимать крышку корпуса - эти переключатели располагаются в легкодоступном месте.

Обычно используются две группы переключателей - DIP Switch 1 и DIP Switch 2. Приведем назначение первой группы переключателей для принтера Epson FX-850/1050:

Переключатель	*Назначение*
SW 1-1	Используемый набор символов: ON используются внутренние шрифты принтера; OFF набор символов может быть задан из программы
SW 1-2	Форма символа "ноль": ON ноль перечеркнут; OFF ноль не перечеркнут
SW 1-3	Выбор таблицы символов: ON используются символы псевдографики; OFF используются символы курсива
SW 1-4	Тип протокола: ON эмуляция IBM Proprinter; OFF протокол ESC/P (для принтеров Epson)
SW 1-5	Пропуск перформации на бумаге: ON не используется; OFF используется
SW 1-6, 1-7, 1-8	Эти переключатели задают национальный набор символов

Переключатель SW 1-1 необходимо установить в положение OFF в том случае, когда Вам необходимо загружать собственный набор символов. Например, в Вашем принтере может не быть набора русских символов. В этом случае Вам надо использовать спе-

циальные программы - русификаторы принтеров. Они будут работать только при правильной установке этого переключателя.

Если Вы установите переключатель SW 1-2 в положение ON, все нули в распечатке будут перечеркнуты. Это удобно для распечатки программ, но не всегда приемлемо для документов.

Если Вы используете псевдографику, переключатель SW 1-3 должен быть установлен в ON, в противном случае вместо символов псевдографики в распечатке появятся латинские буквы.

Тип протокола определяется используемым для печати программным обеспечением. Принтер Epson FX-1050/850 может эмулировать систему команд принтера IBM Proprinter. В приложении приведены команды для протоколов Epson и IBM.

Пропуск перфорации нужен при использовании непрерывной бумажной ленты с перфорацией для отрыва отдельных листов.

Если Ваш принтер содержит набор русских символов, переключатели SW 1-6...1-8 должны быть установлены соответствующим образом. Если Ваш принтер не печатает русские буквы, проверьте правильность установки этих переключателей.

Приведем назначение второй группы переключателей:

Переключатель	*Назначение*
SW 2-1	Длина страницы: ON 12 дюймов; OFF 11 дюймов
SW 2-2	Использование автоматического податчика бумаги: ON используется; OFF не используется
SW 2-3	Использование пропуска для перфорации в 1 дюйм: ON используется; OFF не используется
SW 2-4	Автоматический перевод строки при получении символа возврата каретки: ON используется; OFF не используется

Для других принтеров назначение и нумерация переключателей может отличаться от описанного выше, но их назначение, как правило, аналогично.

Некоторые принтеры, например Epson LQ-2550, не имеют переключателей режимов. Для задания режимов используется клавиатура и небольшой дисплей на корпусе принтера. Режим такого принтера хранится в КМОП-памяти, установленной в принтере и питающейся от аккумулятора. Поэтому установленный режим не сбрасывается при выключении питания принтера.

7.6. Программирование режимов принтера

Для изменения режимов работы принтера и выполнения загрузки шрифтов используются специальные командные последовательности символов. Командные последовательности посылаются в принтер как обычные символы. Вы можете использовать описанные ранее функции MS-DOS или BIOS для вывода этих последовательностей.

Признак начала командной последовательности символов - байт ESC с кодом 1Bh. Вслед за этим байтом программа посылает в принтер саму командную последовательность. Длина последовательности зависит от выполняемой команды. Первый байт командной последовательности - код выполняемой команды. Далее следует один или несколько байтов параметра команды. Некоторым командам не предшествует байт ESC (это, например, команды перевода строки, страницы или команды табуляции).

Подробное описание всех команд не входит в задачу данной книги. В приложении, однако, приведены полные таблицы команд для принтеров Epson LQ-2550 и Epson FX-1050/850 с краткими пояснениями для каждой команды. Мы опишем подробно лишь несколько команд принтера Epson FX-850/1050 с целью иллюстрации способов программирования с использованием протокола ESC/P.

```
ESC '@'     Инициализация принтера
```

Для сброса принтера в исходное состояние программа должна послать на принтер два байта - байт ESC (1Bh) и байт, соответствующий ASCII-символу '@' (40h).

```
07h  Генерация звукового сигнала
```

Если послать этот байт, принтер издаст звуковой сигнал. Сигнал удобно использовать для привлечения внимания оператора, например когда кончилась бумага.

```
0Dh      Возврат каретки
```

Распечатываются все символы из буфера принтера, затем каретка (печатающая головка) возвращается к началу строки. В зависимости от переключателя конфигурации SW 2-4 может дополнительно выполняться прогон бумаги на одну строку.

```
0Ah      Перевод строки
```

Когда этот символ посылается на принтер, все символы, находящиеся во внутреннем буфере принтера, распечатываются, затем каретка возвращается к началу строки и происходит подача листа вперед на одну строку.

```
0Ch      Перевод страницы
```

Принтер распечатывает все символы, находившиеся в буфере, затем выполняет прогон одного листа бумаги.

```
ESC "x" n      Выбор качества печати:
               0 - низкое качество;
               1 - качественный шрифт NLQ.
```

Для задания типа шрифта надо вывести на принтер три байта: символ ESC (1Bh), символ 'x' (78h), затем код шрифта (30h...31h).

Существуют различные команды, позволяющие определить размер межстрочного интервала, расположение левой и правой границ листа, используемый для печати шрифт. Можно выполнять печать графических изображений, о чем мы расскажем позже.

Если Вас не устраивает шрифт, который записан в ПЗУ принтера (например, в нем нет русских букв), Вы можете использовать команды для загрузки собственного шрифта.

Приведем пример программы, которая посылает в принтер командные последовательности и обычные символы, пользуясь функцией 05h прерывания INT 21h:

```
#include <dos.h>
#include <stdio.h>
```

```
main() {
      char buffer[] = {
          0x1b, '@',      // Сбрасываем принтер в исходное
                          // состояние.
          7,7,7,          // Выдаем 3 раза звуковой сигнал.
          0x1b, 'x','0', // Устанавливаем низкое
                          // качество печати.
          'S','t','r','i','n','g',' ','1',  // Печатаем
                                            // строку.
          0x1b, 'x','1', // Устанавливаем высокое
                          // качество печати.
          'S','t','r','i','n','g',' ','2',  // Печатаем
                                            // строку.
          0x0a,           // Переводим строку.
          7,7,7,          // Выдаем 3 раза звуковой сигнал.
          0
      };
      char *p;
// Выводим строку символов на принтер
       for(p = buffer; *p != 0; p++) bdos(0x05, *p, 0);
}
```

Для вывода символа на принтер через функцию MS-DOS здесь использована функция bdos(), входящая в состав стандартных библиотек трансляторов Microsoft QC 2.5 и C 6.0. Первый параметр функции bdos() - номер выполняемой функции прерывания MS-DOS INT 21h, второй - содержимое регистра DX перед вызовом этой функции, и третий - содержимое регистра AL.

В комментариях к программе объясняется назначение управляющих последовательностей, посылаемых на принтер. Перед запуском программы необходимо убедиться в том, что принтер включен и находится в состоянии ON LINE, иначе программа перейдет в состояние ожидания.

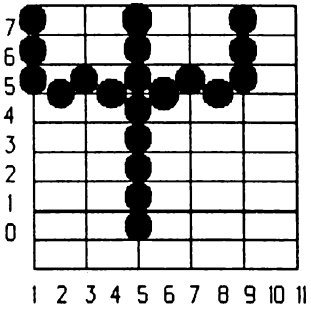
7.7. Печать русских букв

Если среди национальных наборов символов, имеющихся в постоянном запоминающем устройстве, имеются русские буквы, то

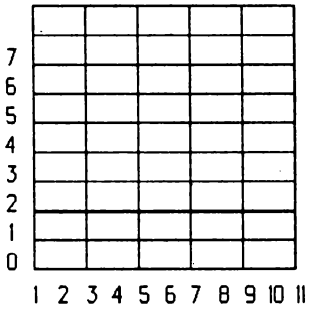
Вам достаточно правильно установить переключатели SW 1-6...1-8 и SW 1-1.

Если же набора русских символов нет или в ПЗУ принтера использована не та кодировка русских символов, Вам потребуются специальные программы загрузки шрифтов, такие, как SETPR, комплекс SOLO, LOADFONT или аналогичные. Все эти программы используют специальные командные последовательности для переопределения тех символов, коды которых соответствуют русским буквам.

Для разработки собственных символов используется сетка. В 9-иголочных принтерах она имеет 11 столбцов и 9 строк:



Сетка, использующая 8 верхних иголок печатающей головки



Сетка, использующая 8 нижних иголок печатающей головки

Из 9 строк может использоваться только 8 верхних или 8 нижних (это показано на левом и правом рисунках соответственно).

Обычно символ располагается выше утолщенной линии, т. е. в строках с номерами от 1 до 7. Исключение составляют такие буквы, как "у", "ц" и т. п. Нижние "хвостики" этих букв должны находиться на строке с номером 0.

Есть ограничение на расположение отдельных точек определяемого символа в строке - точки не должны находиться рядом, т. е. между ними должна находиться одна свободная ячейка.

Для переопределения символов 9-иголочный принтер Epson использует команду ESC '&':

```
ESC '&' '0' n1 n2 a1 d1 d2 ... dn        Определить символы
```

Параметры n1 и n2 задают диапазон кодов ASCII символов, начертание которых необходимо переопределить. Если Вы переопределяете только один символ, эти два параметра должны быть одинаковыми.

Параметр a1 определяет ширину символа в точках и его положение в сетке (использует ли символ верхние восемь линий либо нижние восемь линий). Ширина определяемого символа требуется для печати в пропорциональном режиме, когда место, занимаемое каждой буквой в строке распечатки, зависит от ее ширины. Например, буква "ш" шире, чем буква "и".

Старший бит параметра a1 задает расположение символа в сетке. Если этот бит равен 1, используются восемь верхних линий сетки, если 0 - восемь нижних.

Младшие семь битов задают ширину символа и представляют собой число, определяемое по следующей схеме:

- возмите в качестве начального значения для ширины символа число 8;
- для каждого пустого столбца в сетке с правой стороны символа надо вычесть из начального значения единицу;
- для каждого пустого столбца в сетке с левой стороны символа надо прибавить к начальному значению число 16.

Пусть определяемый символ располагается в верхней части сетки (использует восемь верхних строк). Пусть этот символ начинается в третьем столбце и заканчивается в 7 столбце. Тогда десятичное значение параметра a1 вычисляется таким образом:

```
a1 = 8(начальное значение) -
     - 2(два пустых столбца справа) +
     + 32(два пустых столбца слева) +
     + 128(старший бит равен 1) = 166
```

Если Ваш символ использует верхние восемь строк сетки, начинается в первом столбце и заканчивается в девятом, в качестве параметра a1 подходит значение 136. При этом символы будут печататься верхними восемью иголками печатающей головки. Для использования нижних восьми иголок и такой же ширины символа задайте значение a1, равное 8.

Параметры d1...dn - образцы столбцов точек для определяемого символа. Их должно быть всегда 11, даже если символ содер-

жит пустые столбцы. Для пустых столбцов в качестве образца надо задать 0.

Для включения определенного программой набора символов в работу необходимо выдать команду ESC '%' '0', для использования набора символов из внутреннего ПЗУ принтера выдайте команду ESC '%' '1'.

Приведем пример программы, изменяющей начертание символа '@' в принтере Epson FX-850/1050. Для правильной работы программы переключатель SW 1-1 должен быть установлен в положение OFF.

```
#include <dos.h>
#include <stdio.h>

main() {

      char buffer[] = {
          0x1b, '@', // Сбрасываем принтер в исходное состояние.

          7,7,7,     // Выдаем 3 раза звуковой сигнал.

// Определяем вместо '@' новый символ:

          0x1b, '&', 0,

          '@', '@', 136,
          32,80,168,84,42,84,168,80,32,0,0,

// Выдаем строку символов, используем начертание,
// заданное в ПЗУ принтера.

          '@', '@', '@', '@', '@', 0x0a,

// Используем новое начертание:

          0x1b, '%', 1,
          '@', '@', '@', '@', '@', 0x0a,

// Возвращаемся опять к старому начертанию:

          0x1b, '%', 0,
          '@', '@', '@', '@', '@', 0x0a,

          7,7,7,   // Выдаем 3 раза звуковой сигнал.

          '$'      // Признак конца массива данных

      };

      char *p;

// Выводим строку символов на принтер

           for(p = buffer; *p != '$'; p++)
                  bdos(0x05, *p, 0);
}
```

Рассмотрим теперь метод переопределения начертания символов в 24-иголочном принтере Epson LQ-2550. Для определения начертания символов в этом принтере используется сетка высотой 24 точки - соответственно по одной точке для каждой иголки в печатающей головке. 24-иголочный принтер использует несколько наборов символов:

- черновой набор символов (Draft);
- качественный набор символов (Letter Quality);
- пропорциональный набор символов (Proportional).

В зависимости от используемого набора символов ширина сетки может быть либо 9 точек (для чернового набора символов), либо 29 точек (для качественного набора символов), либо 37 точек (для пропорционального набора символов). Кроме того, для последних двух наборов столбцы сетки расположены ближе друг к другу, чем для чернового набора. На рисунке показаны сетки для чернового и качественного/пропорционального наборов символов принтера Epson LQ-2550 (см. рисунок на следующей странице).

Обычные символы располагаются между жирными линиями. Нижние линии сетки используются для подчеркивания и изображения "хвостиков" таких букв, как "у", "ц" и т. п.

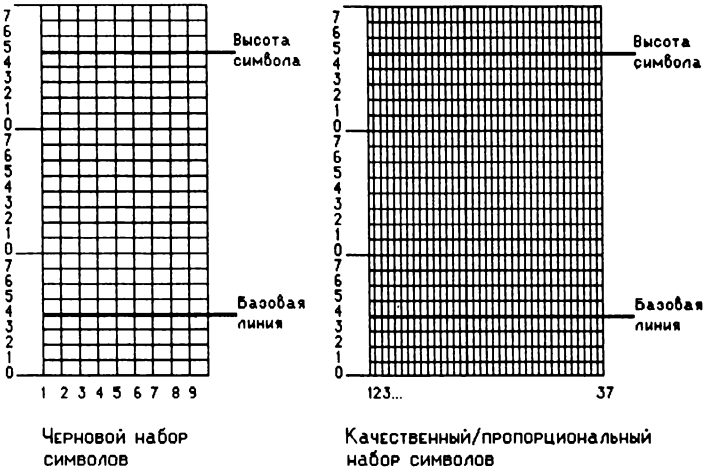
Так же как и для 9-иголочных принтеров, существует ограничение на расположение точек в узлах сетки: справа и слева от каждой точки должны располагаться пустые позиции.

Приведем формат команды для переопределения символов в принтере Epson LQ-2550:

```
ESC '&' '0' n1 n2 d0 d1 d2 data          Определить символы
```

Параметры n1 и n2 задают диапазон кодов ASCII символов, начертание которых необходимо переопределить. Их назначение такое же, как и для 9-иголочных принтеров Epson. Если Вы переопределяете только один символ, эти два параметра должны быть одинаковыми.

Далее следуют три байта данных, которые задают ширину символа и размер свободного пространства вокруг символа. Параметр d0 задает количество свободных столбцов слева, параметр d2 определяет количество свободных столбцов справа от символа. Параметр d1 определяет ширину символа в столбцах сетки.



Изменяя ширину символа и размер свободного пространства вокруг него можно формировать пропорциональные наборы символов. В следующей таблице приведены максимальные значения для параметров d0, d1, d2 для различных наборов символов:

Набор	*d1*	*d0+d1+d2*
Черновой	9	12
Качественный, 10 символов на дюйм	29	36
Качественный, 12 символов на дюйм	23	30
Пропорциональный	37	42

После параметра d2 следует последовательность байтов, описывающих символ, т. е. образец для символа. Для задания одного столбца сетки требуется три байта, поэтому для определения одного символа Вы должны задать (d1 * 3) байтов данных.

7.8. Печать в графическом режиме

До сих пор мы использовали принтер только для печати символьной информации - буквы, цифры, символы псевдографики и т. д. Коды символов мы тем или иным способом выводили на

нтер, принтер распечатывал их с использованием внутренних шрифтов, записанных в ПЗУ принтера, или с использованием загружаемых шрифтов.

Однако печать символов - это не все, на что способен матричный принтер. Вспомним, как происходит печать букв.

Принтерная головка содержит 9 или 24 иголки, расположенных вертикально. Печатающая головка движется в горизонтальном направлении вдоль листа бумаги, причем между головкой и листом находится красящая лента. Каждая иголка управляется отдельным электромагнитом, при включении которого она "выстреливается" в направлении красящей ленты, оставляя на бумаге маленькую точку. Цвет этой точки определяется красящей лентой. Цветные матричные принтеры (например, LQ2550) заряжаются многоцветной лентой. Эта лента напоминает черно-красную ленту для механических пишущих машинок, но на ней больше цветов. Обычно используются четырехцветные ленты, раскрашенные следующими цветами - желтый, красный, синий, черный.

В процессе печати кассета с цветной лентой может перемещаться в вертикальном направлении, поэтому перед печатающей головкой может оказаться участок ленты с одним из цветов.

При наложении цветов возможно получение различных цветовых комбинаций, поэтому, хотя цветов на ленте всего четыре, цветной принтер может печатать достаточно качественные и многоцветные изображения.

Когда принтер печатает алфавитно-цифровой символ, он, передвигая печатающую головку слева направо (или справа налево, что тоже возможно), "выстреливает" теми иголками, которые соответствуют заданному шрифтом изображению символа.

Очевидно, что, если бы у нас была возможность самостоятельно управлять иголками и перемещением кассеты с цветной красящей лентой, мы могли бы самостоятельно формировать цветное графическое изображение, так, как это делается в полиграфии. Если Вы внимательно посмотрите на фотографии, напечатанные в газетах или журналах, Вы заметите, что графическое изображение состоит из отдельных точек разного цвета и очень маленького размера.

Практически любые матричные принтеры позволяют печатать графические изображения. Девятииголочные принтеры Epson FX

используют для графической печати 8 верхних иголок. При этом за один проход можно напечатать одну графическую "строку", послав в принтер битовый образ строки.

Если печатаемое графическое изображение по высоте превышает 8 точек, оно выводится в несколько приемов, построчно. После вывода очередной строки программа должна продвинуть бумагу на одну строку вперед. Для того чтобы между графическими строками не оставалось свободного места, необходимо правильно установить межстрочный интервал - не более 8/72 дюйма.

Как перевести принтер в режим графической печати? Для этого принтер Epson FX-1050 использует следующую команду:

```
ESC '*' m n1 n2 data          Печать в графическом режиме
```

В этой команде m задает режим печати:

Значение m	*Режим*
0	Одинарная плотность, 60 точек на дюйм
1	Двойная плотность, 120 точек на дюйм
2	Двойная плотность, печать с высокой скоростью, 120 точек на дюйм
3	Учетверенная плотность, 240 точек на дюйм
4	Режим CRT I, плотность 80 точек на дюйм
5	Режим плоттера (1:1), плотность 72 точки на дюйм
6	Режим CRT II, плотность 90 точек на дюйм
7	Режим плоттера с двойной плотностью, 144 точки на дюйм

Параметры n1 и n2 определяют длину печатаемой графической строки в точках. При определении длины графической строки необходимо учитывать, что в режиме одинарной плотности на строке длиной 8 дюймов можно разместить 480 точек, в режиме учетверенной плотности - около 2000.

Так как передача данных в принтер выполняется по байтам, для представления длины строки приходится использовать два байта информации. Для вычисления параметров n1 и n2 можно пользоваться следующей схемой действий:

- делим длину строки на 256, целочисленный результат деления используем в качестве параметра n2;

- остаток от деления используем в качестве n1.

Например, пусть нам надо распечатать строку из 1234 точек. Тогда параметр n2 будет равен 1234 / 256 = 4. Остаток от деления составит 1234 - 256 * 4 = 210. Это и есть параметр n1.

```
Проверяем: 4 * 256 + 210 = 1234
```

Команда должна всегда содержать два параметра, даже если параметр n2 получился равным нулю.

Вслед за параметрами n1 и n2 должны следовать байты графических данных, предназначенные для печати. Должно быть передано точно n2 * 256 + n1 байтов.

Если Вы передадите меньше графических данных, чем это было определено в команде ESC '*', следующие вводимые в принтер команды или данные будут интерпретироваться как графические данные. Если Вы передадите больше графических данных, чем нужно, лишние данные будут напечатаны как обычный текст.

Для представления одного восьмиточечного столбца графической строки используется один байт данных: верхней точке в столбце соответствует старший разряд байта, а нижней - младший:

*	7	Этому столбцу соответствует
o	6	байт 10011011b или 9Bh
o	5	
*	4	
*	3	
o	2	
*	1	
*	0	

На этом рисунке '*' означает точку в столбце графической строки, 'o' - отсутствие точки.

Вам необходимо подготовить таким образом байты для всех столбцов, входящих в распечатываемую графическую строку.

Цветной 24-иголочный принтер Epson LQ-2550 может работать в описанном выше 8-битовом графическом режиме. Это сделано для обеспечения совместимости со старым программным обеспечением, рассчитанным на принтеры серий FX, RX, LX и EX. Однако все возможности этого принтера раскрываются только при использовании всех его 24 иголок. В этом случае графическое

изображение печатается отдельными строчками, высота которых составляет 24 точки. При этом для представления одного столбца графической строки требуется три байта данных. Каждый байт должен готовиться отдельно, при этом можно считать, что 24-битовая графическая строка состоит из трех 8-битовых.

Формат команды графической печати для этого принтера расширен по сравнению с описанным выше:

```
ESC '*' m n1 n2 data          Печать в графическом режиме
```

В этой команде m, как и раньше, задает режим печати. Однако для этого параметра определено больше значений:

Значение m	*Режим*
0	Одинарная плотность, 60 точек на дюйм, 8-битовая графика
1	Двойная плотность, 120 точек на дюйм, 8-битовая графика
2	Двойная плотность, печать с высокой скоростью, 120 точек на дюйм, 8-битовая графика
3	Учетверенная плотность, 240 точек на дюйм, 8-битовая графика
4	Режим CRT I, плотность 80 точек на дюйм, 8-битовая графика
6	Режим CRT II, плотность 90 точек на дюйм, 8-битовая графика
32	Одинарная плотность, 60 точек на дюйм, 24-битовая графика
33	Двойная плотность, 120 точек на дюйм, 24-битовая графика
38	Режим CRT III, плотность 90 точек на дюйм, 24-битовая графика
39	Тройная плотность, 180 точек на дюйм, 24-битовая графика
40	Шестикратное увеличение плотности, 360 точек на дюйм, 24-битовая графика

Учтите, что для команды 24-битового графического вывода требуется массив графических данных в три раза больше по размеру, чем для 8-битовой команды.

Приведем пример программы, выводящей на принтер в режиме 8-битовой графики строку, состоящую из 40 столбцов:

```
#include <dos.h>
#include <stdio.h>

union REGS rg;

int main() {

     int i;

// Переводим строку

     printchar(0x0d);
     printchar(0x0a);

// Выдаем на принтер команду графической
// печати (ESC "*" m n1 n2 data)

     printchar(27);
     printchar('*');

// Выбираем режим 0 - 8-битовая графика,
// одинарная плотность

     printchar(0);

// Задаем длину графической строки:
//    n1 = 40; n2 = 0

     printchar(40);
     printchar(0);

// Выводим в цикле 40 раз байт DBh

     for(i=0; i<40; i++)
          printchar(0xdb);

// Переводим строку

     printchar(0x0d);
     printchar(0x0a);
}

// ------------------------------------
// Эта функция выводит один символ
// на стандартный принтер (LPT1)
// ------------------------------------

int printchar(int chr) {

// Вызываем функцию 5 прерывания INT 21h -
// распечатка символа на принтере.
```

```
        rg.h.ah = 5;
        rg.h.dl = chr;
        int86(0x21, &rg, &rg);
}
```

Аналогичная программа, использующая 24-битовую графику на принтере Epson LQ-2550:

```
#include <dos.h>
#include <stdio.h>

union REGS rg;

int main() {

     int i;

// Переводим строку

     printchar(0x0d);
     printchar(0x0a);

// Выдаем на принтер команду графической
// печати (ESC '*' m n1 n2 data)

     printchar(27);
     printchar('*');

// Выбираем режим 32 - 24-битовая графика,
// одинарная плотность

     printchar(32);

// Задаем длину графической строки:
//    n1 = 40; n2 = 0

     printchar(40);
     printchar(0);

// Выводим в цикле 120 раз байт DBh. Для вывода
// строки из 40 столбцов в 24-битовом режиме
// требуется в три раза больше графических данных,
// чем для 8-битового режима.

     for(i=0; i<120; i++)
          printchar(0xdb);

// Переводим строку

     printchar(0x0d);
     printchar(0x0a);
}

// ------------------------------------
// Эта функция выводит один символ
// на стандартный принтер (LPT1)
// ------------------------------------
```

```
int printchar(int chr) {
// Вызываем функцию 5 прерывания INT 21h -
// распечатка символа на принтере.
          rg.h.ah = 5;
          rg.h.dl = chr;
          int86(0x21, &rg, &rg);
}
```

Для вывода сложных графических изображений Ваша программа должна сначала подготовить массив данных для построчной 8-битовой или 24-битовой печати. Затем готовый массив можно вывести на принтер, используя несколько команд графического вывода (по одной команде на одну графическую строку).

Глава 8

КОНТРОЛЛЕР ПРЕРЫВАНИЯ

В первой книге первого тома мы уже рассказывали о контроллере прерываний и о механизме прерываний в персональном компьютере IBM PC/XT/AT. Для полноты изложения мы приведем этот материал здесь еще раз, т. к. контроллер прерываний занимает одно из центральных мест в архитектуре персонального компьютера. Без умения работать с контроллером прерываний Вы не сможете использовать режим прямого доступа к памяти, который будет обсуждаться в этой книге.

8.1. Механизм прерываний

Для обработки событий, происходящих асинхронно по отношению к выполнению программы, лучше всего подходит механизм прерываний. Прерывание можно рассматривать как некоторое особое событие в системе, требующее моментальной реакции. Например, хорошо спроектированные системы повышенной надежности используют прерывание по аварии в питающей сети для выполнения процедур записи содержимого регистров и оперативной памяти на магнитный носитель, с тем чтобы после восстановления питания можно было продолжить работу с того же места.

Кажется очевидным, что возможны самые разнообразные прерывания по самым различным причинам. Поэтому прерывание рассматривается не просто как таковое, с ним связывают число, называемое номером типа прерывания или просто номером прерывания. С каждым номером прерывания связывается то или иное событие. Система умеет распознавать, какое прерывание, с каким номером произошло, и запускает соответствующую этому номеру процедуру.

Программы могут сами вызывать прерывания с заданным номером. Для этого они используют команду INT. Это так называемые программные прерывания. Программные прерывания не являются асинхронными, т. к. вызываются из программы (а она-то знает, когда она вызывает прерывание!).

Программные прерывания удобно использовать для организации доступа к отдельным, общим для всех программ модулям. Например, программные модули операционной системы доступны прикладным программам именно через прерывания, и нет необходимости при вызове этих модулей знать их текущий адрес в памяти. Прикладные программы могут сами устанавливать свои обработчики прерываний для их последующего использования другими программами. Для этого встраиваемые обработчики прерываний должны быть резидентными в памяти. Мы научимся создавать свои программы обработки прерываний и будем говорить об этом при обсуждении резидентных программ.

Аппаратные прерывания вызываются физическими устройствами и приходят асинхронно. Эти прерывания информируют систему о событиях, связанных с работой устройств, например о том, что завершилась печать символа на принтере и неплохо было бы выдать следующий символ, или о том, что требуемый сектор диска уже прочитан, его содержимое доступно программе.

Использование прерываний при работе с медленными внешними устройствами позволяют совместить ввод/вывод с обработкой данных в центральном процессоре и в результате повышает общую производительность системы.

Некоторые прерывания (первые пять в порядке номеров) зарезервированы для использования самим центральным процессором на случай каких-либо особых событий вроде попытки деления на ноль, переполнения и т. п.

Иногда желательно сделать систему нечувствительной ко всем или отдельным прерываниям. Для этого используют так называемое маскирование прерываний, о котором мы еще будем подробно говорить. Но некоторые прерывания замаскировать нельзя, это немаскируемые прерывания.

Заметим еще, что обработчики прерываний могут сами вызывать программные прерывания, например для получения доступа к сервису BIOS или DOS (сервис BIOS также доступен через механизм программных прерываний).

Составление собственных программ обработки прерываний и замена стандартных обработчиков DOS и BIOS является ответственной и сложной работой. Необходимо учитывать все тонкости работы аппаратуры и взаимодействия программного и аппаратно-

го обеспечения. При отладке возможно разрушение операционной системы с непредсказуемыми последствиями, поэтому надо очень внимательно следить за тем, что делает Ваша программа.

8.2. Таблица векторов прерываний

Для того чтобы связать адрес обработчика прерывания с номером прерывания, используется таблица векторов прерываний, занимающая первый килобайт оперативной памяти - адреса от 0000:0000 до 0000:03FF. Таблица состоит из 256 элементов - FAR-адресов обработчиков прерываний. Эти элементы называются векторами прерываний. В первом слове элемента таблицы записано смещение, а во втором - адрес сегмента обработчика прерывания.

Прерыванию с номером 0 соответствует адрес 0000:0000, прерыванию с номером 1 - 0000:0004 и т. д. Для программиста, использующего язык Си, таблицу можно описать так:

```
void (* interrupt_table[256])();
```

Инициализация таблицы происходит частично BIOS после тестирования аппаратуры и перед началом загрузки операционной системой, частично при загрузке DOS. DOS может переключить на себя некоторые прерывания BIOS.

Займемся теперь содержимым таблицы векторов прерываний. Приведем назначение некоторых наиболее важных векторов:

Номер	*Описание*
0	Ошибка деления. Вызывается автоматически после выполнения команд DIV или IDIV, если в результате деления происходит переполнение (например, при делении на 0). DOS обычно при обработке этого прерывания выводит сообщение об ошибке и останавливает выполнение программы. Для процессора 8086 при этом адрес возврата указывает на следующую после команды деления команду, а в процессоре 80286 - на первый байт команды, вызвавшей прерывание

1 Прерывание пошагового режима. Вырабатывается после выполнения каждой машинной команды, если в слове флагов установлен бит пошаговой трассировки TF. Используется для отладки программ. Прерывание не вырабатывается после выполнения команды MOV в сегментные регистры или после загрузки сегментных регистров командой POP

2 Аппаратное немаскируемое прерывание. Это прерывание может использоваться по-разному в разных машинах. Обычно вырабатывается при ошибке четности в оперативной памяти и при запросе прерывания от сопроцессора

3 Прерывание для трассировки. Это прерывание генерируется при выполнении однобайтовой машинной команды с кодом CCh и обычно используется отладчиками для установки точки прерывания

4 Переполнение. Генерируется машинной командой INTO, если установлен флаг OF. Если флаг не установлен, то команда INTO выполняется как NOP. Это прерывание используется для обработки ошибок при выполнении арифметических операций

5 Печать копии экрана. Генерируется при нажатии на клавиатуре клавиши PrtScr. Обычно используется для печати образа экрана. Для процессора 80286 генерируется при выполнении машинной команды BOUND, если проверяемое значение вышло за пределы заданного диапазона

6 Неопределенный код операции или длина команды больше 10 байт (для процессора 80286)

7 Особый случай отсутствия математического сопроцессора (процессор 80286)

8 IRQ0 - прерывание интервального таймера, возникает 18,2 раза в секунду

9	IRQ1 - прерывание от клавиатуры. Генерируется при нажатии и при отжатии клавиши. Используется для чтения данных от клавиатуры
A	IRQ2 - используется для каскадирования аппаратных прерываний в машинах класса AT
B	IRQ3 - прерывание асинхронного порта COM2
C	IRQ4 - прерывание асинхронного порта COM1
D	IRQ5 - прерывание от контроллера жесткого диска для XT
E	IRQ6 - прерывание генерируется контроллером флоппи-диска после завершения операции
F	IRQ7 - прерывание принтера. Генерируется принтером, когда он готов к выполнению очередной операции. Многие адаптеры принтера не используют это прерывание
10	Обслуживание видеоадаптера
11	Определение конфигурации устройств в системе
12	Определение размера оперативной памяти в системе
13	Обслуживание дисковой системы
14	Последовательный ввод/вывод
15	Расширенный сервис для AT-компьютеров
16	Обслуживание клавиатуры
17	Обслуживание принтера
18	Запуск BASIC в ПЗУ, если он есть
19	Загрузка операционной системы
1A	Обслуживание часов
1B	Обработчик прерывания Ctrl-Break
1C	Прерывание возникает 18,2 раза в секунду, вызывается рограммно обработчиком прерывания таймера
1D	Адрес видеотаблицы для контроллера видеоадаптера 6845

1E	Указатель на таблицу параметров дискеты
1F	Указатель на графическую таблицу для символов с кодами ASCII 128-255
20-5F	Используется DOS или зарезервировано для DOS
60-67	Прерывания, зарезервированные для пользователя
68-6F	Не используются
70	IRQ8 - прерывание от часов реального времени
71	IRQ9 - прерывание от контроллера EGA
72	IRQ10 - зарезервировано
73	IRQ11 - зарезервировано
74	IRQ12 - зарезервировано
75	IRQ13 - прерывание от математического сопроцессора
76	IRQ14 - прерывание от контроллера жесткого диска
77	IRQ15 - зарезервировано
78-7F	Не используются
80-85	Зарезервированы для BASIC
86-F0	Используются интерпретатором BASIC
F1-FF	Не используются

IRQ0-IRQ15 - это аппаратные прерывания, о них будет рассказано позже.

8.3. Маскирование прерываний

Часто при выполнении критических участков программ, для того чтобы гарантировать выполнение определенной последовательности команд, целиком приходится запрещать прерывания. Это можно сделать командой CLI. Ее нужно поместить в начало критической последовательности команд, а в конце расположить команду STI, разрешающую процессору воспринимать прерыва-

ния. Команда CLI запрещает только маскируемые прерывания, немаскируемые всегда обрабатываются процессором.

Если Вы используете запрет прерываний с помощью команды CLI, следите за тем, чтобы прерывания не отключались на длительный период времени, т. к. это может привести к нежелательным последствиям, например будут отставать часы.

Если Вам надо запретить не все прерывания, а только некоторые, например от клавиатуры, то для этого надо воспользоваться услугами контроллера прерываний. Подробно об этом немного ниже, сейчас отметим только, что выдачей в этот контроллер определенной управляющей информации можно замаскировать прерывания от отдельных устройств.

8.4. Изменение таблицы векторов прерываний

Вашей программе может потребоваться организовать обработку некоторых прерываний. Для этого программа должна переназначить вектор на свой обработчик. Это можно сделать, изменив содержимое соответствующего элемента таблицы векторов прерываний.

Очень важно не забыть перед завершением работы восстановить содержимое измененных векторов в таблице прерываний, т. к. после завершения работы программы память, которая была ей распределена, считается свободной и может быть использована для загрузки другой программы. Если Вы забыли восстановить вектор и пришло прерывание, то система может разрушиться - вектор теперь указывает на область, которая может содержать что угодно.

Поэтому последовательность действий для нерезидентных программ, желающих обрабатывать прерывания, должна быть такой:

- прочитать содержимое элемента таблицы векторов прерываний для вектора с нужным Вам номером;
- запомнить это содержимое (адрес старого обработчика прерывания) в области данных программы;
- установить новый адрес в таблице векторов прерываний так, чтобы он соответствовал началу Вашей программы обработки прерывания;

- перед завершением работы программы прочитать из области данных адрес старого обработчика прерывания и записать его в таблицу векторов прерываний.

Кроме того, операция изменения вектора прерывания должна быть непрерывной в том смысле, что во время изменения не должно произойти прерывание с номером, для которого производится замена программы обработки. Если Вы, например, запишете новое значение смещения, а сегментный адрес обновить не успеете, то по какому адресу будет передано управление в случае прерывания и что при этом произойдет? Об этом можно только догадываться.

Для облегчения работы по замене прерывания DOS предоставляет в Ваше распоряжение специальные функции для чтения элемента таблицы векторов прерывания и для записи в нее нового адреса. Если Вы будете использовать эти функции, DOS гарантирует, что операция по замене вектора будет выполнена правильно. Вам не надо заботиться о непрерывности процесса замены вектора прерывания.

Для чтения вектора используйте функцию 35h прерывания 21h. Перед ее вызовом регистр AL должен содержать номер вектора в таблице. После выполнения функции в регистрах ES:BX будет искомый адрес обработчика прерывания.

Функция 25h прерывания 21h устанавливает для вектора с номером, находящимся в AL, обработчик прерывания DS:DX.

Разумеется, Вы можете непосредственно обращаться к таблице векторов прерываний, но тогда при записи необходимо замаскировать прерывания командой CLI, не забыв разрешить их после записи командой STI.

Для пользователей языка Си библиотека Quick C содержит функции _dos_getvec(), _dos_setvect(). Первая функция получает адрес из таблицы векторов прерываний, вторая устанавливает новый адрес.

Если Вам надо добавить какие-либо собственные действия к тем, что выполняет стандартный обработчик прерывания, то можно организовать цепочку прерываний.

Для организации цепочки прерываний Вы записываете в векторную таблицу адрес своего обработчика, не забыв сохранить

прежнее содержимое таблицы. Ваш обработчик получает управление по прерыванию, выполняет какие-либо действия, затем передает управление стандартному обработчику. Можно сделать и по-другому: Ваш обработчик вызывает стандартный как подпрограмму, после возврата из стандартного обработчика выполняет дополнительные действия, т. е. Вы можете вставить дополнительную обработку как до вызова стандартного обработчика, так и после его вызова.

В библиотеке Quick C имеется функция для организации цепочки прерываний - _chain_intr().

Рассмотрим более подробно возможности библиотеки интегрированной среды Quick C, предназначенные для работы с прерываниями.

Модификатор interrupt (_interrupt для Quick C 2.5 и C 6.0) описывает функцию, которая является обработчиком прерывания. Такая функция завершается командой возврата из обработки прерывания IRET, и для нее автоматически генерируются команды сохранения регистров на входе и их восстановления при выходе из обработчика прерывания. Пример использования модификатора для описания функции обработки прерывания:

```
void interrupt far int_funct(void) {
    // Тело обработчика прерывания
}
```

Функция обработки прерывания должна быть FAR-функцией, т. к. таблица векторов прерываний содержит полные адреса в виде сегмент:смещение.

Ключевое слово interrupt используется также для описания переменных, предназначенных для хранения векторов прерываний:

```
void (_interrupt _far *oldvect)(void);
```

Модификаторы _interrupt и _far для Quick C 2.5 и C 6.0 являются синонимами соответственно interrupt и far.

Какие требования предъявляются к программе обработки прерывания?

Если прерывания происходят часто, то их обработка может сильно замедлить работу прикладной программы. Поэтому обработчик прерывания должен быть короткой быстро работающей программой, которая выполняет только самые необходимые дей-

ствия. Например, считать очередной символ из порта принтера и поместить его в буфер, увеличить значение какого-либо глобального счетчика прерываний и т. п.

Для установки своего обработчика прерываний используйте функцию _dos_setvec. Эта функция имеет два параметра - номер прерывания и указатель на новую функцию обработки прерывания. Например:

```
_dos_setvect(0x16, my_key_intr);
```

В этом примере для клавиатурного прерывания с номером 16h устанавливается новый обработчик прерывания my_key_intr.

Если Вам надо узнать адрес старого обработчика прерывания по его номеру, лучше всего воспользоваться функцией _dos_getvect, которая принимает в качесте параметра номер прерывания и возвращает указатель на соответствующий этому номеру в таблице векторов прерываний обработчик. Например:

```
old_vector = _dos_getvect(0x16);
```

Для организации цепочки прерываний используйте функцию _chain_intr. В качестве параметра эта функция принимает адрес старого обработчика прерываний.

Следующий простой пример иллюстрирует применение всех трех функций, предназначенных для работы с прерываниями. Эта программа встраивает собственный обработчик прерывания таймера, который будет вызываться примерно 18,2 раза в секунду. Встраиваемый обработчик прерывания считает тики таймера, и, если значение счетчика кратно 20, на динамик компьютера выдается звуковой сигнал. В конце работы новая программа обработки прерывания таймера вызывает старый обработчик с помощью функции _chain_intr.

После установки нового обработчика прерывания таймера основная программа ждет нажатия любой клавиши клавиатуры, затем она восстанавливает старое содержимое вектора прерывания.

```
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>

// Выключаем проверку стека и указателей
#pragma check_stack( off )
```

```
#pragma check_pointer( off )

// Это макро используется для выдачи
//   сигнала на внутренний динамик
//   компьютера. Используется вывод
//   в формате TTY символа BELL (7)
//   через прерывание BIOS 10h
#define BEEP() _asm { \
                        _asm mov bx,0       \
                        _asm mov ax, 0E07h \
                        _asm int 10h        \
                    }

void main(void);

// Объявление программы обработки прерывания
void _interrupt _far timer(void);
// Эта переменная предназначена для хранения
//   старого значения вектора прерывания
//   таймера. Она должна быть глобальной.
void (_interrupt _far *oldvect)(void);
// Переменная для подсчета тиков таймера
volatile long ticks;

void main(void) {
     ticks=0L;  // Сбрасываем счетчик тиков таймера
     oldvect = _dos_getvect(0x1c); // Запоминаем адрес
                                   // старого обработчика
                                   // прерывания
     _dos_setvect(0x1c, timer);    // Устанавливаем свой
                                   // обработчик
     printf("\nТаймер установлен. Нажмите любую"
               "клавишу...\n");
     getch(); // Ожидаем нажатия на любую клавишу
     _dos_setvect(0x1c,oldvect);   // Восстанавливаем старый
                                   // обработчик прерывания
                                   // таймера
     exit(0);
}
// Функция обрабатывает прерывания таймера
void _interrupt _far timer(void) {
     ticks++; // Увеличиваем счетчик тиков таймера
```

```
    // Если значение счетчика тиков кратно 20,
    // выдаем сигнал на динамик компьютера
    if((ticks % 20) == 0) BEEP();
    // Вызываем старый обработчик прерывания
    _chain_intr(oldvect);
}
```

8.5. Особенности обработки аппаратных прерываний

Аппаратные прерывания вырабатываются устройствами компьютера, когда возникает необходимость их обслуживания. Например, по прерыванию таймера соответствующий обработчик прерывания увеличивает содержимое ячеек памяти, используемых для хранения времени. В отличие от программных прерываний, вызываемых запланированно самой прикладной программой, аппаратные прерывания всегда происходят асинхронно по отношению к выполняющимся программам. Кроме того, может возникнуть одновременно сразу несколько прерываний.

Чтобы система "не растерялась", решая, какое прерывание обслуживать в первую очередь, существует специальная схема приоритетов. Каждому прерыванию назначается свой уникальный приоритет. Если происходит одновременно несколько прерываний, то система отдает предпочтение самому высокоприоритетному, откладывая на время обработку остальных прерываний.

Система приоритетов реализована на двух микросхемах Intel 8259 (для машин класса XT - на одной такой микросхеме). Каждая микросхема обслуживает до восьми приоритетов. Микросхемы можно объединять (каскадировать) для увеличения количества уровней приоритетов в системе.

Уровни приоритетов обозначаются сокращенно IRQ0 - IRQ15 (для машин класса XT существуют только уровни IRQ0 - IRQ7).

Для машин XT приоритеты линейно зависели от номера уровня прерывания. IRQ0 соответствовало самому высокому приоритету, за ним шли IRQ1, IRQ2, IRQ3 и т. д. Уровень IRQ2 в машинах класса XT был зарезервирован для дальнейшего расширения системы. И начиная с машин класса AT IRQ2 стал использоваться для каскадирования контроллеров прерывания 8259. Доба-

вленные приоритетные уровни IRQ8 - IRQ15 в этих машинах располагаются по приоритету между IRQ1 и IRQ3.

Приведем таблицу аппаратных прерываний, расположенных в порядке приоритета:

Номер	*Описание*
8	IRQ0 - прерывание интервального таймера, возникает 18,2 раза в секунду
9	IRQ1 - прерывание от клавиатуры. Генерируется при нажатии и при отжатии клавиши. Используется для чтения данных с клавиатуры
A	IRQ2 - используется для каскадирования аппаратных прерываний в машинах класса AT
70	IRQ8 - прерывание от часов реального времени
71	IRQ9 - прерывание от контроллера EGA
72	IRQ10 - зарезервировано
73	IRQ11 - зарезервировано
74	IRQ12 - зарезервировано
75	IRQ13 - прерывание от математического сопроцессора
76	IRQ14 - прерывание от контроллера жесткого диска
77	IRQ15 - зарезервировано
B	IRQ3 - прерывание асинхронного порта COM2
C	IRQ4 - прерывание асинхронного порта COM1
D	IRQ5 - прерывание от контроллера жесткого диска для XT
E	IRQ6 - прерывание генерируется контроллером флоппи-диска после завершения операции
F	IRQ7 - прерывание принтера. Генерируется принтером, когда он готов к выполнению очередной операции. Многие адаптеры принтера не используют это прерывание

Из таблицы видно, что самый высокий приоритет у прерываний от интервального таймера, затем идет прерывание от клавиатуры.

Для управления схемами приоритетов необходимо знать внутреннее устройство контроллера прерываний 8259. Поступающие

прерывания запоминаются в регистре запроса на прерывание IRR. Каждый бит из восьми в этом регистре соответствует прерыванию. После проверки на обработку в настоящий момент другого прерывания запрашивается информация из регистра обслуживания ISR. Перед выдачей запроса на прерывание в процессор проверяетя содержимое восьмибитового регистра маски прерываний IMR. Если прерывание данного уровня не замаскировано, то выдается запрос на прерывание.

Наиболее интересными с точки зрения программирования контроллера прерываний являются регистры маски прерываний IMR и управляющий регистр прерываний.

В машинах класса XT регистр маски прерываний имеет адрес 21h, управляющий регистр прерываний - 20h. Для машин AT первый контроллер 8259 имеет такие же адреса, что и в машинах XT, регистр маски прерываний второго контроллера имеет адрес A1h, управляющий регистр прерываний - A0h.

Разряды регистра маски прерываний соответствуют номерам IRQ. Для того чтобы замаскировать аппаратное прерывание какого-либо уровня, надо заслать в регистр маски байт, в котором бит, соответствующий этому уровню, установлен в 1. Например, для маскирования прерываний от НГМД в порт 21h надо заслать двоичное число 01000000.

Приведем пример программы, маскирующей прерывание от флоппи-диска:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

void main(void);

void main(void) {

        outp(0x21,0x40);
        printf("\nПрерывания от флоппи-диска запрещены.\n");

        exit(0);
}
```

Эта программа есть на дискете, прилагающейся к книге. Запустите ее (с жесткого диска) и попробуйте поработать, например, с дисководом A:. У вас ничего не получится! Чтобы "оживить" флоппи-диски, запустите программу, которая размаскирует все прерывания (в том числе и от флоппи):

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

void main(void);

void main(void) {

     outp(0x21,0);
     printf("\nПрерывания от флоппи-диска разрешены.\n");

     exit(0);
}
```

Заметьте, что мы только что замаскировали прерывание именно от флоппи-диска, все остальные устройства продолжали нормально работать. Если бы мы выдали машинную команду CLI, то отключились бы все аппаратные прерывания. Это привело бы, например, к тому, что клавиатура была бы заблокирована.

Еще одно замечание, касающееся обработки аппаратных прерываний. Если Вы полностью заменяете стандартный обработчик аппаратного прерывания, не забудьте в конце программы выдать байт 20h в порт с адресом 20h (A0h для второго контроллера 8259). Эти действия необходимы для очистки регистра обслуживания прерывания ISR. При этом разрешается обработка прерываний с более низким приоритетом, чем то, которое только что обрабатывалось.

Если Вы обрабатываете прерывание 1Ch, то добавка в конце программы не нужна, т. к. это прерывание является расширением другого прерывания (прерывания таймера).

Перед тем как завершить изучение прерываний, зададимся вопросом - можно ли замаскировать немаскируемое прерывание? Оказывается, можно!

Конечно, если сигнал прерывания пришел на вход немаскируемого прерывания процессора, ничего сделать нельзя - прерывание произойдет неизбежно. Но в компьютерах XT и AT предусмотрены схемы, блокирующие вход немаскируемого прерывания процессора NMI.

Для XT маскированием немаскируемого прерывания управляет порт с адресом 0A0h. Если записать в него 0, немаскируемое прерывание будет запрещено, если 80h разрешено.

Аналогично для AT маскированием немаскируемого прерывания управляет бит 7 порта 70h. Запись байта 0ADh в порт 70h

запретит немаскируемое прерывание, а байта 2Dh разрешит прохождение прерывания.

Заметим, что мы не запрещаем немаскируемое прерывание "внутри" процессора - это невозможно по определению, мы "не пускаем" сигнал прерывания на вход NMI.

8.6. Контроллер прерываний 8259

Программируемый контроллер прерываний 8259 (отечественный аналог - КР1810ВН59А) предназначен для обработки до восьми приоритетных уровней прерываний. Возможно каскадирование микросхем, при этом общее число уровней прерываний будет достигать 64.

Контроллер 8259 имеет несколько режимов работы, которые устанавливаются программным путем. В персональных компьютерах XT и AT за первоначальную установку режимов работы микросхем 8259 отвечает BIOS. У программиста скорее всего не возникнет потребность перепрограммировать контроллер - это небезопасно, т. к. неправильное программирование контроллера приведет к нарушению логики работы всей системы.

Однако часто возникает необходимость изменения текущего режима работы (запрет или разрешение прерываний определенного или всех уровней, обработка конца прерывания) или опроса состояния внутренних регистров контроллера. Для этого необходимо ознакомиться со справочными данными на микросхему 8259, где детально описано как первоначальное программирование контроллера, так и управление им во время работы.

Каждому приоритетному уровню прерывания микросхема ставит в соответствие определенный, задаваемый программно, номер прерывания. В разделе книги, посвященном особенностям обработки аппаратных прерываний, приводится такое соответствие для машин типа XT и AT.

Если контроллеры 8259 каскадированы, то ведомой микросхеме присваивается код (выдачей в микросхему соответствующего командного слова). Этот код равен номеру входа IRQ ведущей микросхемы, с которым соединен выход запроса прерывания INT ведомой микросхемы. Внутри микросхемы приоритет зависит от номера IRQ и задается программно. Для компьютеров XT и AT

самым высоким приоритетом внутри группы, обслуживаемой каждым контроллером, является вход IRQ0. Однако возможно программное изменение приоритетов в рамках так называемого приоритетного кольца. При этом дно приоритетного кольца имеет самый низкий приоритет. Приведем возможные варианты задания приоритетов:

Вход	*Уровни приоритета*
IRQ0	7 6 5 4 3 2 1 0
IRQ1	0 7 6 5 4 3 2 1
IRQ2	1 0 7 6 5 4 3 2
IRQ3	2 1 0 7 6 5 4 3
IRQ4	3 2 1 0 7 6 5 4
IRQ5	4 3 2 1 0 7 6 5
IRQ6	5 4 3 2 1 0 7 6
IRQ7	6 5 4 3 2 1 0 7

Наиболее высокий приоритет у входа IRQ с обозначением 0 приоритетного кольца, наиболее низкий - с обозначением 7.

Для обработки прерываний контроллер имеет несколько внутренних регистров. Это регистр запросов прерываний IRR, регистр обслуживания прерываний ISR, регистр маски прерываний IMR. В регистре IRR хранятся запросы на обслуживание прерываний от аппаратуры. После выработки сигнала прерывания центральному процессору соответствующий разряд регистра ISR устанавливается в единичное состояние, что блокирует обслуживание всех запросов с равным или более низким приоритетом. Устранить эту блокировку можно либо сбросом соответствующего бита в ISR, либо командой специального маскирования.

Имеется два типа команд, посылаемых программой в контроллер 8259, - команды инициализации и команды операции. Возможны следующие операции:

- индивидуальное маскирование запросов прерывания;
- специальное маскирование обслуженных запросов;
- установка статуса уровней приоритета (по установке исходного состояния, по обслуженному запросу, по указанию);
- операции конца прерывания (обычный конец, специальный конец, автоматический конец);

- чтение регистров IRR, ISR, IMR.

Мы не будем подробно описывать команды инициализации контроллера 8259, т. к. программистам они скорее всего не понадобятся. Желающих разобраться во всех тонкостях задания начального режима работы контроллера прерываний мы отсылаем к справочной литературе по микросхеме 8259 или ее отечественному аналогу.

Рассмотрим команды операций. Они бывают трех типов:

- Маскирование запросов прерывания.
- Команды обработки конца прерывания.
- Опрос регистров и специальное маскирование.

Байты команды маскирования запросов прерывания выводятся соответственно в порты 21h и A1h для первого и второго контроллера 8259 компьютера AT. Команды операций второго и третьего типа используют порты с адресами 20h и A0h.

Маскирование запросов прерываний мы уже описывали в главе, посвященной прерываниям. Для маскирования какого-либо уровня прерывания надо записать в регистр маски IMR по адресу 21h или A1h единицу в соответствующий разряд регистра.

Приведем команды обработки конца прерывания:

Биты байта команды								*Описание*
D7	D6	D5	D4	D3	D2	D1	D0	
0	0	1	0	0	0	0	0	Обычный конец прерывания
0	1	1	0	0	B2	B1	B0	Специальный конец прерывания, B0...B2 - двоично-десятичный код сбрасываемого разряда в регистре обслуживания прерывания ISR
1	0	1	0	0	X	X	X	Циклический сдвиг уровней приоритета с обычным концом прерывания. Дно приоритетного кольца устанавливается по обслуженному запросу

1	1	1	0	0	B2	B1	B0	Циклический сдвиг уровней приоритета со специальным концом прерывания, B0...B2 - двоично-десятичный код дна приоритетного кольца
1	0	0	0	0	X	X	X	Разрешение вращения уровней приоритета
0	0	0	0	0	X	X	X	Сброс разрешения вращения уровней приоритета
1	1	0	0	0	B2	B1	B0	Циклический сдвиг уровней приоритета без завершения прерывания, B0...B2 - двоично-десятичный код дна приоритетного кольца

Команды третьего типа выдаются также в порты с адресами 20h и A0h. Они имеют следующий формат:

Биты байта команды								*Описание*
D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	1	1	X	X	Установка режима опроса
0	0	0	0	1	0	1	1	Разрешение чтения регистра ISR
0	0	0	0	1	0	1	0	Разрешение чтения регистра IRR
0	1	1	0	1	0	0	0	Разрешение триггера специального маскирования
0	1	0	0	1	0	0	0	Сброс триггера специального маскирования

По команде обычного конца прерывания устанавливается в нулевое состояние разряд ISR, соответствующий последнему обслуженному запросу.

Команда специального конца прерывания устанавливает в нулевое состояние тот разряд ISR, номер которого указан в разрядах B0...B2 команды.

Команда циклического сдвига уровней приоритета с обычным концом прерывания устанавливает в ноль разряд ISR, соответствующий последнему обслуженному запросу и этому же номеру запроса присваивается низший уровень приоритета.

Аналогично работает команда циклического сдвига уровней приоритета со специальным концом прерывания, только низший уровень приоритета присваивается тому входу IRQ, номер которого указан в разрядах B0...B2 команды.

Команда циклического сдвига уровней приоритета устанавливает статус уровней без выполнения операции конца прерывания. Разряды B0...B2 указывают дно приоритетного кольца.

После выполнения команд разрешения чтения регистров ISR или IRR при выполнении команды ввода из порта 20h и A0h считывается соответственно содержимое регистров ISR и IRR. Для получения содержимого регистра IMR необходимо выполнить чтение портов с адресами соответственно 21h и A1h.

Команда разрешения триггера специального маскирования блокирует действие тех разрядов ISR, которые замаскированы командой типа 1 (маскирования индивидуальных приоритетных уровней запроса прерывания). Специальное маскирование используется для обслуживания такого запроса, который блокируется старшим или равным по уровню приоритета обслуженным запросом, хранящимся в ISR, не сбрасывая последний.

Чтение регистров ISR и IRR может использоваться резидентными программами при проверке возможности своей активизации - можно проверить, не выполняется ли в настоящий момент обработка какого-нибудь прерывания, которая может конфликтовать с действиями резидентной программы.

ОГЛАВЛЕНИЕ