БИБЛИОТЕКА СИСТЕМНОГО ПРОГРАММИСТА

А. В. ФРОЛОВ, Г. В. ФРОЛОВ

АППАРАТНОЕ ОБЕСПЕЧЕНИЕ IBM PC

ТОМ **2**

ЧАСТЬ 2

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. 1.—2-е изд., стер.—М.: "ДИАЛОГ-МИФИ", 1992.—208 с.—(Библиотека системного программиста; Т. 2)
ISBN 5-86404-025-8 (Т. 2, ч. 2)

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

Во второй части подробно описан контроллер прямого доступа к памяти, арифметический сопроцессор. Большое внимание уделено использованию расширенной и дополнительной памяти. 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 2404090000-010 / 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 2

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 1202.

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

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

© А.В.Фролов, Г.В. Фролов, 1992
© Оригинал-макет, оформление обложки. АО "ДИАЛОГ-МИФИ", 1992

ISBN 5-86404-025-8 (Т. 2, ч. 2)
ISBN 5-86404-004-5

Глава 9

КОНТРОЛЛЕР ПРЯМОГО ДОСТУПА К ПАМЯТИ

Прямой доступ к памяти (Direct Memory Access - DMA) используется для выполнения операций передачи данных непосредственно между оперативной памятью и устройствами ввода/вывода. Обычно это такие устройства, как НГМД, НМД, кассетные накопители на магнитной ленте КНМЛ (стриммеры).

При использовании DMA процессор не участвует в операциях ввода/вывода, контроллер прямого доступа сам формирует все сигналы, необходимые для обмена данными с устройством. Скорость такого непосредственного обмена значительно выше, чем при традиционном вводе/выводе с использованием центрального процессора и команд INP, OUT.

Мы уже немного рассказывали о контроллере прямого доступа к памяти в третьей книге первого тома, в разделе, посвященном работе с НГМД на уровне команд ввода/вывода. Была приведена программа, использующая DMA для чтения секторов дискеты. В этом разделе мы подробнее рассмотрим порты контроллера DMA.

Распространены два типа контроллеров DMA - для IBM PC/XT и для IBM AT. Начнем с первого типа контроллеров, а затем займемся контроллером DMA компьютера IBM AT.

9.1. Контроллер прямого доступа для IBM PC/XT

Контроллер прямого доступа для IBM PC/XT реализован на базе микросхемы Intel 8237A и содержит четыре канала. Эти каналы используются следующим образом:

0 обновление содержимого динамической памяти компьютера, этот канал имеет наивысший приоритет;
1 не используется;
2 адаптер накопителя на гибком магнитном диске (НГМД);
3 адаптер накопителя на магнитном диске (НМД) - этот канал имеет низший приоритет.

9.1.1. Регистры каналов DMA

Каждый канал содержит 16-разрядные регистры:

- регистр текущего адреса CAR, содержит текущий адрес ячейки памяти при выполнении операции обмена данными с использованием DMA;
- регистр циклов прямого доступа к памяти CWR, содержит число слов, предназначенных для передачи минус единица; при выполнении обмена данными регистр работает в режиме вычитания;
- регистр хранения базового адреса BAR, используется для хранения базового адреса памяти, используемого при передаче данных; в процессе работы канала DMA содержимое этого регистра не изменяется;
- регистр хранения базового числа циклов прямого доступа к памяти WCR; он хранит число циклов DMA, его содержимое также не изменяется;
- регистр режима MR, определяющий работу канала.

Приведем адреса регистров и их форматы для компьютеров IBM PC/XT.

Порты 00h - 07h

Эти регистры содержат базовые адреса и счетчики передаваемых данных каналов 0 - 3. Их назначение следующее:

00h	Запись:	Базовый адрес канала 0
	Чтение:	Текущий адрес
01h	Запись:	Счетчик канала 0
	Чтение:	Текущий адрес
02h	Запись:	Базовый адрес канала 1
	Чтение:	Текущий адрес
03h	Запись:	Счетчик канала 1
	Чтение:	Текущий адрес
04h	Запись:	Базовый адрес канала 2
	Чтение:	Текущий адрес
05h	Запись:	Счетчик канала 2
	Чтение:	Текущий адрес

06h	Запись:	Базовый адрес канала 3
	Чтение:	Текущий адрес
07h	Запись:	Счетчик канала 3
	Чтение:	Текущий адрес

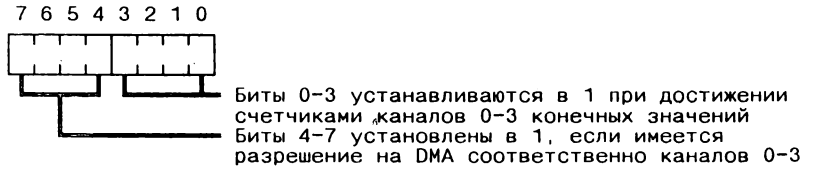
Порт 08h

Этот порт используется при записи в качестве управляющего регистра и при чтении как регистр состояния.

Формат управляющего регистра:

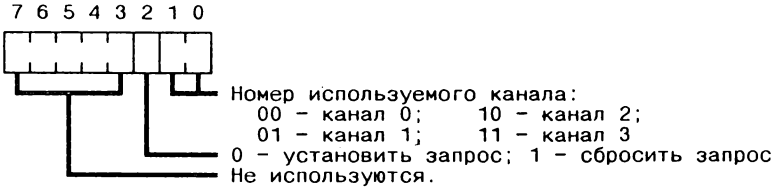
Обычно этот регистр инициализируется BIOS в процессе тестирования системы, и впоследствии изменять режим работы контроллера DMA не требуется. Ошибки при инициализации этого порта могут привести к "зависанию" системы.

При чтении из порта 08h программа получает слово состояния контроллера DMA:



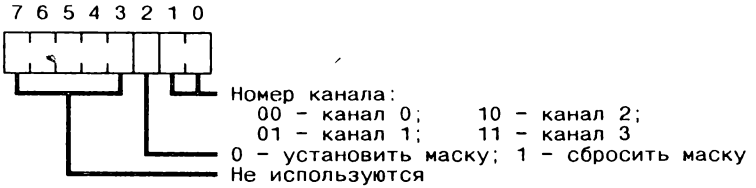
Порт 09h

Регистр запроса. Предназначен для организации программного (а не аппаратного) запроса на DMA. Для использования программного запроса канал должен быть запрограммирован в режиме блочной передачи. Формат регистра:



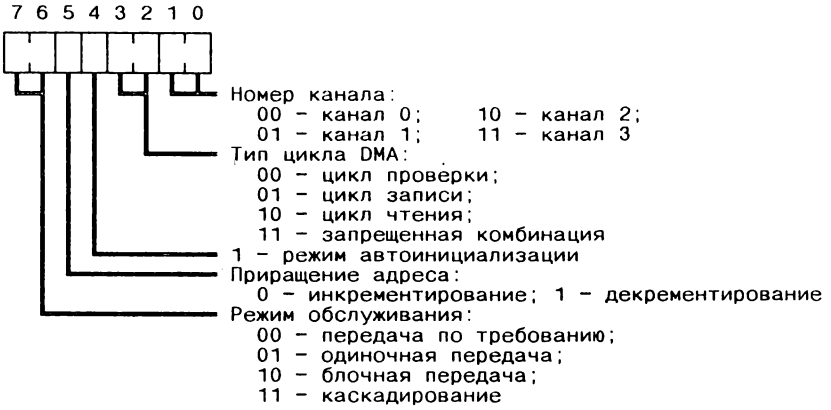
Порт 0Ah

Регистр маски. Используется для маскирования запросов на прямой доступ для отдельных каналов:



Порт 0Bh

Регистр режима. Служит для определения режимов работы каналов контроллера DMA:



Порт 0Ch

Сброс триггера байтов. Для загрузки внутренних 16-разрядных регистров контроллера используется последовательный вывод младшего, затем старшего байтов слова. После сброса триггера байтов можно начинать загрузку 16-разрядных регистров.

Порт 0Dh

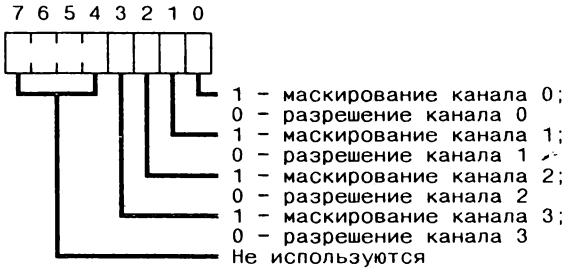
Запись в этот порт вызывает сброс контроллера. Для дальнейшего использования контроллер должен быть заново проинициализирован.

Порт 0Eh

Сброс регистра маски. После записи в этот регистр любого значения разрешается работа всех четырех каналов прямого доступа.

Порт 0Fh

Маскирование/размаскирование каналов. С помощью этого порта можно выполнить одновременное маскирование или размаскирование нескольких каналов:



Порты 81h-8Fh

Это порты регистров страниц.

Для работы с памятью контроллер прямого доступа использует 20-разрядные физические адреса. Шестнадцать младших битов адреса необходимо записать в регистр базового адреса канала. Старшие четыре бита - биты 16 - 19 - должны быть записаны в соответствующие порты регистров страниц. При инициализации регистров базового адреса и регистра страниц необходимо следить за тем, чтобы в процессе передачи данных не происходил переход

за границу 64 килобайт. Для адресации регистров страниц можно использовать порты:

81h	Регистр страниц канала 2
82h	Регистр страниц канала 3
83h	Регистр страниц канала 1

9.1.2. Инициализация канала DMA

Для инициализации канала программа должна выполнить следующие шаги:

- сбросить триггер байтов командой записи в регистр 0Ch;
- задать режим работы канала, выполнив запись по адресу 0Bh в регистр режима MR;
- заслать младшие 16 битов 20-битового адреса области памяти, которая будет использована для передачи данных, в регистр базового адеса (адрес порта зависит от номера канала: 0-й канал использует адрес 00h, 1-й канал - 02h, 2-й канал - 04h, 3-й канал - 06h);
- заслать номер страницы (старшие 4 бита 20-битового адреса) в регистр страниц 81h;
- загрузить регистр циклов прямого доступа к памяти CWR значением, на 1 меньшим требуемого количества передаваемых байтов (адреса этих портов для каналов 0...3 соответственно 01h, 03h, 05h, 07h);
- разрешить работу канала, выполнив запись в регистр маски каналов по адресу 0Ah.

Сразу после разрешения канал начинает передачу данных. Окончив передачу, устройство обычно вырабатывает прерывание, которое служит признаком окончания передачи данных.

9.2. Контроллер прямого доступа для IBM AT

Контроллер DMA компьютера IBM AT совместим снизу вверх с контролером IBM PC/XT. Он состоит из двух каскадно включенных микросхем Intel 8237A-5. Второй контроллер обслуживает каналы DMA с номерами 4 - 7.

Приведем назначение каналов DMA для IBM AT:

0	Зарезервировано
1	Управление синхронной передачей данных SDLC (Synchronous Data Link Control)
2	Адаптер накопителя на гибком магнитном диске (НГМД)
3	Адаптер накопителя на магнитном диске (НМД);
4	Используется для каскадного соединения с первым контроллером DMA
5-6	Зарезервировано

Другое отличие - это разрядность каналов. Каналы 0 - 3 являются каналами 8-битовой передачи данных, а каналы 4 - 7 обеспечивают 16-битовую передачу данных. В связи с этим используются все 8 битов регистров страниц. Формируется 24-битовый адрес из 16 младших битов адреса, записываемых в базовые регистры, и 8 старших битов адреса, записываемых в регистры страниц.

Размер страницы составляет 128 килобайт, поэтому при передаче данных с использованием DMA не должна пересекаться граница 128 килобайт.

Приведем назначение и адреса регистров страниц контроллера для IBM AT:

81h	Регистр страниц канала	2
82h	-"-	3
83h	-"-	1
87h	-"-	0
89h	-"-	6
8Bh	-"-	5
8Ah	-"-	7
8Fh	Регенерация динамической памяти	

Для 16-битовых каналов 4 - 7 передача данных начинается с границы слова и все адреса относятся к 16-битовым словам.

Порты 0C0h - 0DFh

Эти регистры содержат базовые адреса и счетчики передаваемых данных каналов 4-7. Их назначение следующее:

0C0h	Запись:	Базовый адрес канала 4
	Чтение:	Текущий адрес

0C2h	Запись:	Счетчик канала 4
	Чтение:	Текущий адрес
0C4h	Запись:	Базовый адрес канала 5
	Чтение:	Текущий адрес
0C6h	Запись:	Счетчик канала 5
	Чтение:	Текущий адрес
0C8h	Запись:	Базовый адрес канала 6
	Чтение:	Текущий адрес
0CAh	Запись:	Счетчик канала 6
	Чтение:	Текущий адрес
0CCh	Запись:	Базовый адрес канала 7
	Чтение:	Текущий адрес
0CEh	Запись:	Счетчик канала 7
	Чтение:	Текущий адрес

Порты 0D0h-0DFh

Это управляющие порты и порты состояния второй микросхемы 8237A-5. По формату и назначению они соответствуют рассмотренным ранее для контроллера DMA компьютеров IBM PC/XT:

0D0h	Управляющий регистр/регистр состояния
0D2h	Регистр запроса
0D4h	Регистр маски
0D6h	Регистр режима
0D8h	Сброс триггера байтов
0DAh	Сброс контроллера
0DCh	Сброс регистра маски
0DEh	Маскирование/размаскирование каналов

В качестве примера использования контроллера прямого доступа к памяти приведем программу чтения сектора флоппи-диска. Мы уже описывали ее в предыдущем томе. Поэтому здесь мы не будем описывать команды контроллера НГМД и другие тонкости, имеющие отношение к работе с флоппи-дисками.

Перед началом инициализации КПДП программа посылает в порты 0Bh и 0Ch код операции, которая будет выполняться КПДП: 46h - для операции чтения, и 4Ah - для операции записи.

В процессе инициализации программа должна сообщить КПДП адрес буфера, куда ему следует поместить данные или откуда надо взять данные, и длину передаваемых данных в байтах.

Адрес необходимо представить в виде номера страницы и смещения. Для КПДП машины AT используется 8-битовый номер страницы и 16-битовое смещение. Например, для адреса 23456 номер страницы - 2, смещение - 3456.

Для программирования канала 2 КПДП программа должна сначала вывести младший байт смещения в порт с адресом 4, затем вывести в этот же порт старший байт смещения и, наконец, вывести байт номера страницы в порт с адресом 81h.

Длина передаваемых данных выводится аналогично в порт с адресом 5 - сначала младший байт длины, затем старший.

После определения режима работы канала, адреса буфера и длины передаваемых данных, программа должна разрешить работу КПДП, выдав в порт с адресом 0Ch байт 2. Теперь канал прямого доступа готов к работе и будет ждать данных от контроллера НГМД.

Приведенная ниже демонстрационная программа использует несколько наиболее характерных команд контроллера НГМД. Она предназначена для работы на машине AT. Для того чтобы она правильно работала и на машинах PC/XT, ее надо немного изменить. Изменения касаются программирования контроллера ПДП и программирования скорости передачи контроллера НГМД.

Контроллер КПДП PC/XT использует 4-битовый номер страницы буфера вместо 8-битового. Скорость передачи контроллера НГМД в машинах PC/XT не программируется, надо убрать соответствующие строки из программы. Еще надо обратить внимание на различное быстродействие машин AT и PC/XT и скорректировать константы в строках программы, выполняющих задержку.

Программа не проверяет, установлен ли флоппи-диск в приемный карман дисковода, поэтому перед запуском не забудьте установить диск.

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

```
#define CYL 0

void main(void);
void fdc_out(unsigned char byte);
int  fdc_inp(void);
void int_wait(void);
void dma_init(char *);

void main(void) {

     unsigned i;
     long l;
     char buffer[512];
     char status[7], main_status;
     DPT _far *fdpt;
     FILE *sect;

     printf("\n"
            "\nРабота с контроллером НГМД"
            "\n  (C)Фролов А., 1991"
            "\n");

// Эта программа предназначена только для IBM AT

     if(pc_model() != 0xfc) {
          printf("Эта программа предназначена только для IBM"
                 "AT\n");
          exit(-1);
     }

// Открываем файл, в который будем записывать
// содержимое самого первого сектора на дискете

     sect = fopen("!sector.dat","wb+");

// Устанавливаем указатель на таблицу параметров дискеты

     fdpt = get_dpt();

// Включаем мотор дисковода A:, перед этим разрешаем прерывания

     _enable();
     outp(0x3F2, 0x1C);

// Выполняем задержку для разгона двигателя

     for(l=0;l<200000;l++);

// Показываем содержимое регистра основного
// состояния контроллера

     printf("Мотор включен.\t\t");
     printf("Основное состояние: %02.2X\n",inp(0x3F4));

// Перед чтением сектора необходимо установить головку на нужную
// дорожку, в нашем случае это дорожка с номером CYL.

// Выдаем контроллеру команду "Поиск"

     fdc_out(0xf);
```

```
// Для команды "Поиск" требуется два байта параметров:
// номер головки/номер накопителя и номер дорожки.
// Мы работаем с нулевой головкой накопителя A:,
// поэтому первый параметр равен 0, второй - CYL

     fdc_out(0);
     fdc_out(CYL);

// Показываем содержимое регистра основного
// состояния контроллера

     printf("\n<<<Поиск>>> \t\t");
     printf("Основное состояние: %02.2X\n",inp(0x3F4));

// Ожидаем прерывание по завершении операции

     int_wait();

// Задержка для позиционирования головки

     for(i=0;i<20000;i++);

// Для проверки результата выполнения команды "Поиск" выдаем
// контроллеру команду "Чтение состояния прерывания"

// Выводим содержимое регистра состояния ST0 и номер дорожки
// после выполнения команды "Поиск" PCN

     fdc_out(0x8);
     printf("Состояние прерывания:\t");
     printf(" ST0: %02.2X, \t", fdc_inp());
     printf("PCN: %02.2X\n", fdc_inp());

// Для более глубокой диагностики состояния
// контроллера выдаем контроллеру команду
// "Чтение состояния накопителя", выводим
// содержимое регистра состояния ST3

     fdc_out(4);
     fdc_out(0);
     printf("Состояние накопителя:\t ST3: %02.2X\n",fdc_inp());

// Устанавливаем скорость передачи данных 500 Кбайт/с,
// это значение может различаться для разных типов дискет

     outp(0x3F7, 0);

// Инициализация канала прямого
// доступа к памяти

     dma_init(buffer);

// Выдаем команду "Чтение данных"

     fdc_out(0x66);
     fdc_out(0x0);      // накопитель 0, головка 0

     fdc_out(CYL);      // цилиндр CYL
     fdc_out(0);        // головка 0
     fdc_out(1);        // номер сектора - 1
```

```
// Передаем контроллеру технические параметры
// дисковода, берем их из таблицы параметров дискеты.
// Это такие параметры:
//    - размер сектора;
//    - номер последнего сектора на дорожке;
//    - размер промежутка;
//    - число считываемых/записываемых байтов
     fdc_out(fdpt->sec_size);
     fdc_out(fdpt->eot);
     fdc_out(fdpt->gap_rw);
     fdc_out(fdpt->dtl);
// Ожидаем прерывание по завершении операции
     int_wait();
// Считываем и выводим на экран байты результата
// операции "Чтение данных"
     printf("\n<<<Чтение сектора>>> \n");
     printf("   Байты состояния (ST0,ST1,ST2,C,H,R,N):\n");
     for(i=0; i<7; i++) printf("%02.2X\t", (char) fdc_inp());
     printf("\n");
// Выводим содержимое считанного сектора в файл
     for(i=0; i<512; i++) fputc(buffer[i],sect);
     fclose(sect);
// Выключаем мотор
     outp(0x3F2, 0xC);
}

// Вывод байта в контроллер дисковода
void fdc_out(unsigned char parm) {
     _asm {
          mov   dx,3F4h      // Порт основного состояния
loop_fdc_out:
          in    al,dx
          test  al,80h       // Проверяем готовность
          jz loop_fdc_out    //   контроллера
          inc   dx           // Выводим байт в порт данных
          mov   al, parm     //   контроллера
          out   dx, al
     }
}
// Ввод байта из порта данных контроллера дисковода
int fdc_inp(void) {
```

```
     _asm {
          mov   dx,3F4h      // Порт основного состояния
loop_fdc_inp:
          in    al,dx
          test  al,80h       // Проверяем готовность
          jz loop_fdc_inp    //   контроллера
          inc   dx           // Введенный байт записываем
          in    al, dx       // в регистр AX
     }
}
// Ожидание прерывания от контроллера
void int_wait(void) {
// Разрешаем прерывания
     _enable();
     _asm {
          mov   ax,40h          // После прихода прерывания
          mov   es,ax           // программа обработки прерывания
          mov   bx,3Eh          // устанавливает в 1 старший бит
wait_loop:                      // байта в области данных BIOS
          mov   dl,es:[bx]      // по адресу 0040:003E.
          test  dl,80h          // Мы ждем, когда этот бит будет
          jz    wait_loop       // установлен в 1, а затем
                                // сбрасываем его.
          and   dl,01111111b
          mov   es:[bx],dl
     }
}
// Инициализация канала прямого доступа к памяти
void dma_init(char *buf) {
     unsigned long f_adr;
     unsigned sg, of;
// Вычисляем 24-разрядный адрес буфера для данных
     f_adr = ((unsigned long)_psp << 4)
                  + (((unsigned long)buf) & 0xffff);
// Расщепляем адрес на номер страницы
// и смещение
     sg = (f_adr >> 16) & 0xff;
     of = f_adr & 0xffff;
// На время программирования контроллера прямого
// доступа запрещаем прерывания
     _disable();
```

```
    _asm {
         mov   al,46h    // Команда чтения данных от
                         // контроллера НГМД.
         out   12,al     // Сброс триггера-указателя байта
                         // для работы с 16-разрядными портами.
                         // Следующий байт, выводимый
                         // в 16-разрядный
                         // порт будет интерпретироваться
                         // как младший.
         out   11,al     // Установка режима контроллера ПДП
         mov   ax,of     // Смещение буфера, младший байт
         out   4,al
         mov   al,ah     // Смещение буфера, старший байт
         out   4,al
         mov   ax,sg     // Номер страницы
         out   81h,al
         mov   ax,511    // Длина передаваемых данных
         out   5,al
         mov   al,ah
         out   5,al
         mov   al,2      // Разблокировка канала 2 контроллера
                         // ПДП
         out   10,al
    }
// Инициализация контроллера закончена,
// разрешаем прерывания.
    _enable();
}
```

Остальные команды Вы можете попробовать сами. Для получения дополнительной информации по контроллеру НГМД обратитесь к техническому руководству по IBM PC. Многое можно почерпнуть из описания микросхем дискового контроллера 765 фирмы NEC и аналогов этой микросхемы - Intel 8272A и отечественной КР1810ВГ72А.

На этом мы завершим обсуждение контроллера DMA. Советуем Вам еще раз посмотреть программу, читающую секторы диска с использованием канала прямого доступа в памяти, которую мы приводили в третьей книге первого тома. Вы можете самостоятельно внести в нее некоторые усовершенствования, например проверку перехода адреса в процессе работы канала прямого доступа через границу 128 килобайт.

Глава 10

РАСШИРЕННАЯ ПАМЯТЬ

Компьютеры IBM AT, PS/2 оснащены расширенной памятью, располагающейся в диапазоне адресов свыше одного мегабайта. Однако операционная система MS-DOS, использующая процессоры 80286, 80386 и 80486 в реальном режиме, не имеет полноценного доступа к этой памяти. То же относится и к программам, разработанным для выполнения в среде MS-DOS. Единственное, что MS-DOS версий более ранних, чем 4.0, могла сделать с расширенной памятью, - это разместить там быстродействующий электронный диск, или кэш, накопителя на магнитном диске.

Однако в составе MS-DOS версии 4.0 и более поздних версий появился драйвер расширенной памяти HIMEM.SYS, который в некоторой степени облегчает жизнь программистам, составляющим программы для MS-DOS. Этот драйвер расширяет основное адресное пространство 640 килобайт еще примерно на 64 килобайта и предоставляет относительно удобное средство для хранения в расширенной памяти массивов данных.

Будучи установлен в операционной системе, драйвер HIMEM.SYS предоставляет программам интерфейс в соответствии со спецификацией XMS (eXtended Memory Specification), разработанной корпорациями Lotus, Intel, Microsoft, AST Research.

10.1. Основные понятия

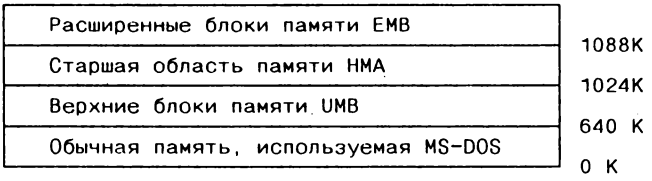
При обсуждении спецификации XMS мы будем использовать следующие понятия и термины.

- Расширенная память (Extended Memory) - это память, используемая в компьютерах с процессорами 80286, 80386, 80486, располагающаяся в адресном пространстве выше одного мегабайта.
- Старшая область памяти (High Memory Area) HMA - это первые 64 килобайта расширенной памяти, начинающиеся с адреса FFFFh:0010h. Адрес конца области HMA - FFFFh:FFFFh. Следовательно, размер области составляет

64 килобайта без 16 байтов. Следует отметить, что эта область может адресоваться процессором в реальном режиме и поэтому может быть использована обычными программами, предназначенными для работы в MS-DOS.

- Верхние блоки памяти (Upper Memory Blocks) UMB - блоки памяти на машинах, использующих процессор 8086. Эти блоки памяти располагаются между границей 640 килобайт и 1 мегабайт. Расположение и размер этих блоков могут сильно изменяться в зависимости от аппаратуры.
- Расширенные блоки памяти (Extended Memory Blocks) EMB - блоки расширенной памяти, располагающиеся выше границы HMA.
- Линия A20 - 21-я адресная линия процессора. Обычно эта линия заблокирована. Разблокировка линии открывает программам доступ к области HMA.

На рисунке схематично показано расположение различных перечисленных выше блоков памяти в адресном пространстве:



10.2. Установка драйвера HIMEM.SYS

Для установки драйвера файл CONFIG.SYS должен содержать строку:

```
device=[d:][путь]himem.sys [/HMAMIN=h] [/NUMHANDLES=n]
```

Параметр /HMAMIN= (необязательный) задает минимальный размер памяти, который могут использовать программы в области HMA. Размер задается в килобайтах. Смысл использования этого параметра заключается в том, чтобы позволять использовать область HMA только тем программам, которые затребуют из этой области не меньше h килобайт. Это нужно для того, чтобы более эффективно использовать область HMA. Если параметр не задан, используется по умолчанию значение 0. Это означает, что первая

же программа, запросившая область HMA, получит к ней доступ. Программа, запущенная следом и, возможно, использующая эту память эффективнее, уже не сможет воспользоваться областью HMA. Максимальное значение параметра h - 63.

Параметр /NUMHANDLES= задает максимальное количество областей расширенной памяти (блоков EMB), которое может быть запрошено программами. Диапазон задаваемых значений - от 1 до 128, значение по умолчанию - 32.

Задавая большие значения n, помните, что на управление каждым блоком EMB тратится 6 байт резидентной памяти.

При установке драйвер HIMEM.SYS может выдавать сообщения об ошибках в следующих случаях:

- используется MS-DOS более старой версии, чем 3.00;
- машина использует процессор 8086, а не 80286, 80386 или 80486;
- при использовании компьютеров с нестандартными схемами управления расширенной памятью и линией A20.

10.3. Спецификация XMS

Спецификация XMS содержит описание программного интерфейса драйвера HIMEM.SYS и рекомендации по использованию области памяти HMA.

10.3.1. Проверка подключения драйвера

Первое, что должна сделать программа, которая собирается вызывать драйвер HIMEM.SYS, - проверить, был ли установлен этот драйвер при загрузке операционной системы.

Для этого надо загрузить в регистр AX значение 4300h и вызвать прерывание INT 2Fh. Если регистр AL будет содержать значение 80h, драйвер установлен, в противном случае - нет. Приведем фрагмент программы, проверяющей подключение драйвера:

```
; Проверяем, установлен ли драйвер HIMEM.SYS

    mov   ax, 4300h
    int   2fh
    cmp   al, 80h
    je    HMM_installed ; Выполняем переход,
                        ; если драйвер установлен
```

10.3.2. Получение адреса управляющей программы

Для вызова драйвера программа должна получить адрес специальной управляющей программы, которая выполняет все функции по обслуживанию расширенной памяти и области HMA.

Этот адрес можно получить, если загрузить в регистр AX значение 4310h и вызвать прерывание INT 2Fh. Прерывание возвратит сегментный адрес управляющей программы в регистре ES, смещение - в регистре BX:

```
; Получаем адрес управляющей функции драйвера

    mov   ax, 4310h
    int   2fh
    mov   word ptr cs:[HMMEntry][0], bx
    mov   word ptr cs:[HMMEntry][2], es
```

В дальнейшем полученный адрес используется для выполнения функций по обслуживанию расширенной памяти. Перед вызовом управляющей программы код требуемой функции должен быть загружен в регистр AH:

```
; Получаем номер версии драйвера HIMEM.SYS

    mov   ax,0
    call  [HMMEntry]
```

Программы, которые обращаются к управляющей функции, должны перед вызовом функции иметь размер стека не менее 256 байтов.

10.3.3. Описание функций драйвера HIMEM.SYS

Все функции драйвера HIMEM.SYS могут быть разделены на следующие пять групп:

- функции получения информации о драйвере (0h);
- функции управления областью HMA (1h...2h);
- функции управления линией A20 (3h...7h);
- функции управления расширенной памятью (8h...Fh);
- функции управления блоками UMB (10h...11h).

Приведем подробное описание этих функций в соответствии со спецификацией XMS версии 2.0.

Получить версию XMS

На входе:	AH	=	00h.
На выходе:	AX	=	номер версии XMS;
	BX	=	номер внутренней модификации драйвера;
	DX	=	0001h - если существует область HMA, 0000h - если область HMA не существует.
Ошибки:	Нет.		

Функция возвращает номера версии и модификации XMS в двоично-десятичном (BCD) формате. Например, если AX=0250h, это означает, что драйвер реализует спецификацию XMS версии 2.50. Дополнительно функция позволяет проверить наличие в системе области HMA.

Запросить область HMA

На входе:	AH	=	01h;
	DX	=	размер памяти в байтах в области HMA, которая будет использоваться резидентными программами или драйверами, обычная программа должна использовать значение DX=FFFFh.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка.
Ошибки:	BL	=	80h, 81h, 90h, 91h, 92h (описание кодов ошибок будет приведено после описания всех функций).

С помощью этой функции программа может зарезервировать для себя область HMA. Задаваемый в регистре DX размер памяти сравнивается с указанным в параметре драйвера /HMAMIN=. Область HMA распределяется запросившей программе только в том случае, если запрошенный в регистре DX размер больше или равен указанному в параметре /HMAMIN. Такой механизм позволяет ограничить использование области HMA только теми программами, которые используют ее наилучшим образом.

Поясним это на примере. Пусть при инициализации операционной системы из файла AUTOEXEC.BAT запускаются две программы. Одна из них использует 10 килобайт из области HMA и запускается первой (в регистре DX функции 01h эта программа указывает значение 10240). Вторая запускаемая программа использует 40 килобайт и запускается после первой. Очевидно, что вторая программа использует область HMA более эффективно. Но т. к. область HMA уже распределена первой программе, вторая программа не сможет ее использовать.

Задавая параметр /HMAMIN=40, мы запретим распределение области HMA тем программам, которые используют в ней меньше 40 килобайт. Теперь первая программа не получит доступ к области HMA, даже если она будет запускаться до второй, использующей 40 килобайт памяти из области HMA.

Освободить область HMA

На входе:	AH	=	02h.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка.
Ошибки:	BL = 80h, 81h, 90h, 93h.		

Программы, которые запрашивали область HMA, должны освободить ее с использованием этой функции. При этом данные, которые находились в этой области, будут потеряны.

После того как программа освободила область HMA, эта область становится доступной другим программам.

Глобальное открывание линии A20

На входе:	AH	=	03h.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка.
Ошибки:	BL	=	80h, 81h, 82h.

Эта функция предназначена для тех программ, которые будут использовать область HMA. Она разрешает работу заблокированной по умолчанию 21-й адресной линии процессора. Перед воз-

вратом управления системе программа должна закрыть линию A20 с помощью функции 04h.

Следует отметить, что на многих типах компьютеров переключение линии A20 достаточно медленная операция.

Глобальное закрывание линии A20

На входе:	AH	=	04h.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка.
Ошибки:	BL	=	80h, 81h, 82h, 94h.

Функция предназначена для тех программ, которые используют область HMA. Она должна выполняться перед завершением работы такой программы.

Локальное открывание линии A20

На входе:	AH	=	05h.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка.
Ошибки:	BL	=	80h, 81h, 82h.

Эта функция предназначена только для тех программ, которые непосредственно управляют расширенной памятью. Перед завершением работы программа должна закрыть линию A20 при помощи функции 06h.

Локальное закрывание линии A20

На входе:	AH	=	06h.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка.
Ошибки:	BL	=	80h, 81h, 82h, 94h.

Функция отменяет разрешение линии A20, запрошенное предыдущей функцией. Она предназначена только для тех программ, которые непосредственно управляют расширенной памятью.

Определение состояния линии A20

На входе:	AH	=	07h.
На выходе:	AX	=	0001h - если линия A20 открыта; 0000h - если линия A20 закрыта.
Ошибки:	BL	=	00h, 80h, 81h.

Функция выполняет попытку адресоваться за границу 1 мегабайта памяти и проверяет, не происходит ли при этом обращение в начало памяти (т. е. "свертка памяти").

Определение размера свободной расширенной памяти

На входе:	AH	=	08h.
На выходе:	AX	=	размер наибольшего свободного блока расширенной памяти в килобайтах;
	DX	=	общий размер свободной расширенной памяти в килобайтах.
Ошибки:	BL	=	80h, 81h, A0h.

При определении размера свободной расширенной памяти в возвращаемое значение не включается 64 килобайта области HMA, даже если эта область не используется программами.

Получить блок EMB

На входе:	AH	=	09h.
	DX	=	размер требуемого блока в килобайтах.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка;
	DX	=	16-битовый индекс (handle) полученного блока EMB.
Ошибки:	BL	=	80h, 81h, A0h, A1h.

Функция заказывает блок EMB из пула свободной расширенной памяти. При успешном выполнении запроса функция возвращает индекс полученного блока, который должен использоваться программой для выполнения всех операций с блоком EMB. Если блок EMB программе больше не нужен, она должна освободить его с помощью функции 0Ah.

Количество блоков EMB, которое может быть заказано, определяется в командной строке драйвера HIMEM.SYS параметром /NUMHANDLES=. Значение по умолчанию - 32, максимальное значение - 128.

Освободить блок EMB

На входе:	AH	=	0Ah;
	DX	=	16-битовый индекс (handle) полученного блока EMB.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка.
Ошибки:	BL	=	80h, 81h, A2h, B2h.

Функция освобождает блок EMB, заказанный предыдущей функцией. При этом все данные, находившиеся в блоке, будут потеряны.

Копирование блоков EMB

На входе:	AH	=	Bh.
	DS:SI	=	указатель на управляющую структуру, определяющую, откуда, куда и как будет выполняться копирование.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка.
Ошибки:	BL	=	80h, 81h, 82h, A3h, A4h, A5h, A6h, A7h, A8h, A9h.

Управляющая структура:

```
ExtMemMoveStruct     struc

 Length           dd ?      ; количество пересылаемых байтов

 SourceHandle     dw ?      ; индекс исходного блока
 SourceOffset     dd ?      ; смещение в исходном блоке
 DestHandle       dw ?      ; индекс блока-назначения
 DestOffset       dd ?      ; смещение в блоке-назначении

ExtMemMoveStruct     ends
```

Эта функция выполняет основную операцию с блоками EMB - копирование данных. Данные могут пересылаться между обычной памятью и блоками EMB, между различными блоками EMB и даже внутри обычной памяти.

Поле Length управляющей структуры указывает количество пересылаемых байтов. Это количество должно быть четным.

Поля SourceHandle и DestHandle указывают соответственно индексы исходного и результирующего блоков EMB. Если в качестве индекса задано значение 0000h, это означает, что в качестве источника или приемника данных используется обычная память.

Поля SourceOffset и DestOffset указывают 32-битовое смещение в блоке EMB или адрес в обычной памяти. В последнем случае этот адрес имеет стандартный формат сегмент:смещение.

Функция копирования сама управляет линией A20, восстанавливая ее состояние после выполнения копирования. Поэтому программе не требуется управлять линией A20.

Во время выполнения копирования разрешены прерывания.

Блокирование EMB

На входе:	AH	=	0Ch;
	DX	=	16-битовый индекс (handle) блокируемого EMB.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка;
	DX:BX	=	32-битовый линейный адрес заблокированного EMB.
Ошибки:	BL	=	80h, 81h, A2h, ACh, ADh.

Функция блокирует EMB и возвращает его базовый адрес как линейный 32-разрядный адрес. Для заблокированного EMB невозможно выполнить операцию копирования. Полученный линейный адрес действителен только для заблокированного EMB.

Разблокирование EMB

На входе:	AH	=	0Dh;
	DX	=	16-битовый индекс (handle) разблокируемого EMB.

На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка.
Ошибки:	BL	=	80h, 81h, A2h, AAh.

Функция разблокирует EMB, заблокированный при вызове предыдущей функции. Полученный от нее линейный адрес становится недействительным.

Получить информацию об индексе EMB

На входе:	AH	=	0Eh;
	DX	=	16-битовый индекс (handle) EMB.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка;
	BH	=	содержимое счетчика блокировок EMB;
	BL	=	количество свободных индексов EMB в системе;
	DX	=	размер блока в килобайтах.
Ошибки:	BL	=	80h, 81h, A2h.

Эта функция используется для получения различной информации об используемых блоках EMB. Линейный адрес блока может быть получен с помощью фунции 0Ch.

Изменить размер EMB

На входе:	AH	=	0Fh;
	DX	=	16-битовый индекс (handle) незаблокированного EMB, размер которого должен быть изменен;
	BX	=	новый размер EMB в килобайтах.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка.
Ошибки:	BL	=	80h, 81h, A0h, A1h, A2h, ABh.

Функция изменяет размер незаблокированного EMB. Если блок уменьшается в размерах, данные в старших адресах блока будут потеряны.

Запросить область UMB

На входе:	AH	=	10h;
	DX	=	размер запрашиваемого блока UMB в параграфах.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка;
	BX	=	сегмент полученного UMB;
	DX	=	размер полученного блока или размер максимального свободного блока UMB (если невозможно выделить блок требуемого размера).
Ошибки:	BL	=	80h, B0h, B1h.

Эта функция позволяет программе получить доступ к блокам UMB, лежащих в пределах первого мегабайта адресного пространства. Для использования этих блоков не требуется управлять линией A20.

Если Вам надо определить размер доступной области UMB, задайте при вызове этой функции DX=FFFFh.

Освободить область UMB

На входе:	AH	=	11h;
	DX	=	сегмент освобождаемого UMB.
На выходе:	AX	=	0001h - если функция выполнена успешно, 0000h - если произошла ошибка.
Ошибки:	BL	=	80h, B2h.

После освобождения блока EMB данные, которые там находились, будут потеряны.

10.3.4. Коды ошибок

Приведем таблицу кодов ошибок, возвращаемых функциями в регистре BL:

Код	*Ошибка*
00h	Нет ошибки, нормальное завершение
80h	Функция не реализована в текущей версии драйвера
81h	Обнаружен драйвер VDISK.SYS, с этим драйвером драйвер HIMEM.SYS несовместим
82h	Ошибка при работе с линией A20
8Eh	Общая ошибка драйвера
8Fh	Катастрофическая ошибка драйвера
90h	Область HMA не существует
91h	Область HMA уже используется
92h	Содержимое регитра DX меньше парметра /HMAMIN=
93h	Область HMA не распределена программе
94h	Линия A20 все еще разблокирована
A0h	Вся расширенная память уже распределена
A1h	Больше нет свободных индексов EMB
A2h	Неправильный индекс EMB
A3h	Неправильный SourceHandle
A4h	Неправильный SourceOffset
A5h	Неправильный DestHandle
A6h	Неправильный DestOffset
A7h	Неправильный Length
A8h	Неразрешенное перекрытие данных при выполнении операции пересылки данных
A9h	Произошла ошибка четности
AAh	EMB не заблокирован
ABh	EMB заблокирован
ACh	Переполнение счетчика блокировок EMB
ADh	Не удалось выполнить блокировку EMB
B0h	Доступен UMB меньшего размера
B1h	Нет доступных блоков UMB
B2h	Задан неправильный сегмент UMB

10.4. Ограничения при использовании области HMA

К сожалению, на программы, использующие область HMA, накладываются значительные ограничения. Они связаны с тем, что MS-DOS версий 4.01 и более ранних, а также BIOS не были рассчитаны на работу с адресами памяти выше границы 1 мегабайт. Приведем список этих ограничений.

- Нельзя передавать MS-DOS FAR-указатели на данные, размещенные в области HMA, т. к. функции MS-DOS проверяют правильность таких указателей.
- Не рекомендуется использование области HMA для выполнения обмена данных с диском через прерывания MS-DOS, BIOS или другими способами.
- Драйверы и резидентные программы, использующие область HMA, не должны держать линию A20 постоянно включенной, т. к. это может привести к неправильной работе программ, не рассчитанных на наличие и доступность этой области.
- Векторы прерываний не должны указывать в область HMA. Это результат предыдущего ограничения.

10.5. Примеры программ

Первая программа демонстрирует проверку подключения драйвера и использование его основных функций:

```
include sysp.inc

        .MODEL  tiny
        DOSSEG

        .STACK  100h

        .DATA

msg     DB 13,10,"Работа с драйвером HIMEM.SYS", 13, 10
        DB "Copyright (C)Frolov A.,1991",13,10,13,10
        DB "$"

noHMM     DB 13,10
          DB "Драйвер HIMEM.SYS не установлен",13,10,"$"
yesHMM    DB 13,10,"Драйвер HIMEM.SYS установлен, ", "$"
ver1      DB "версия: ", "$"
```

```
ver2      DB ", номер модификации: ", "$"
errmsg    DB 13,10,"Ошибка с кодом ", "$"
okmsg     DB 13,10,"Успех!!!", "$"
hmareq    DB 13,10,"Запрашиваем область HMA", "$"
hmarel    DB 13,10,"Освобождаем область HMA", "$"
enA20     DB 13,10,"Открываем линию A20", "$"
dsA20     DB 13,10,"Закрываем линию A20", "$"
loc_enA20 DB 13,10,"Локальный доступ к линии A20","$"
loc_dsA20 DB 13,10,"Закрываем локальный доступ"
          DB      " к линии A20", "$"
check_A20 DB 13,10,"Проверяем доступность "
          DB      "линии A20", "$"
free_ext_mem DB 13,10,"Всего расширенной "
          DB      "памяти, Кбайт: ", "$"
max_ext_block DB 13,10,"Максимальный участок "
          DB      "свободной"
          DB " расширенной памяти, Кбайт: ", "$"

HMMEntry dd ?

        .CODE
        .STARTUP

          mov     ah, 9h ; Выводим заголовок
          mov     dx, OFFSET msg
          int     21h

; Проверяем, установлен ли драйвер HIMEM.SYS

          mov   ax, 4300h
          int   2fh
          cmp   al, 80h
          je    HMM_installed

; Если не установлен, выводим сообщение и завершаем
; работу программы

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

          jmp   terminate

HMM_installed:

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

; Получаем адрес управляющей функции драйвера

          mov   ax, 4310h
          int   2fh
          mov   word ptr cs:[HMMEntry][0], bx
          mov   word ptr cs:[HMMEntry][2], es
```

```
; Получаем номер версии
          mov   ah, 9h
          mov   dx, OFFSET ver1
          int   21h
          mov   ax,0
          call  cs:[HMMEntry]
; Выводим номер версии на экран
          call  Print_word
          mov   ah, 9h
          mov   dx, OFFSET ver2
          int   21h
          mov   ax, bx
          call  Print_word
; Запрашиваем область HMA
          mov   ah, 9h
          mov   dx, OFFSET hmareq
          int   21h
          mov   ax,0100h
          mov   dx,0ffffh
          call  cs:[HMMEntry]
          or    ax, ax
          jnz   hmareq_ok
          jmp   error
hmareq_ok:
          mov   ah, 9h
          mov   dx, OFFSET okmsg
          int   21h
; Открываем линию A20
          mov   ah, 9h
          mov   dx, OFFSET enA20
          int   21h
          mov   ax,0300h
          call  cs:[HMMEntry]
          or    ax, ax
          jnz   enA20_ok
          jmp   error
enA20_ok:
          mov   ah, 9h
          mov   dx, OFFSET okmsg
          int   21h
  Закрываем линию A20
```

```
          mov   ah, 9h
          mov   dx, OFFSET dsA20
          int   21h

          mov   ax,0400h

          call  cs:[HMMEntry]
          or    ax, ax
          jnz   dsA20_ok
          jmp   error

dsA20_ok:

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

; Освобождаем область HMA
          mov   ah, 9h
          mov   dx, OFFSET hmarel
          int   21h

          mov   ax,0200h

          call  cs:[HMMEntry]
          or    ax, ax
          jz    error

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

; Получаем локальный доступ к линии A20
          mov   ah, 9h
          mov   dx, OFFSET loc_enA20
          int   21h

          mov   ax,0500h

          call  cs:[HMMEntry]
          or    ax, ax
          jz    error

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

; Проверяем линию A20
          mov   ah, 9h
          mov   dx, OFFSET check_A20
          int   21h

          mov   ax,0700h

          call  cs:[HMMEntry]
          or    ax, ax
          jz    error
```

```
          mov   ah, 9h
          mov   dx, OFFSET okmsg
          int   21h

; Определяем размер свободной расширенной памяти

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

          mov   ax,0800h

          call  cs:[HMMEntry]

          push  ax
          mov   ax, dx
          call  Print_word

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

          pop   ax
          call  Print_word

; Освобождаем линию A20

          mov   ah, 9h
          mov   dx, OFFSET loc_dsA20
          int   21h

          mov   ax,0600h

          call  cs:[HMMEntry]
          or    ax, ax
          jz    error

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

          jmp   terminate

error:
          push  bx

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

          pop   ax
          call  Print_word

terminate:

            .EXIT   0

; Вывод на экран содержимого регистра AX
```

```
Print_word proc near
;--------------------
       push ax
       push bx
       push dx
;
       push ax
       mov cl,8
       rol ax,cl
       call Byte_to_hex
       mov bx,dx
       @@out_ch bh
       @@out_ch bl
;
       pop ax
       call Byte_to_hex
       mov bx,dx
       @@out_ch bh
       @@out_ch bl
;
       pop dx
       pop bx
       pop ax
       ret
Print_word endp
;
Byte_to_hex proc near
;--------------------
; al - input byte
; dx - output hex
;--------------------
       push ds
       push cx
       push bx

       lea bx,tabl
       mov dx,cs
       mov ds,dx
;
       push ax
       and al,0fh
       xlat
       mov dl,al
;
       pop ax
       mov cl,4
       shr al,cl
       xlat
       mov dh,al
;
```

```
        pop bx
        pop cx
        pop ds
        ret
;
tabl db '0123456789ABCDEF'
Byte_to_hex endp
;
            END
```

Теперь приведем пример программы, использующей область HMA для выполнения процедуры генерации звукового сигнала. Программа получает доступ к области HMA, копирует в нее процедуру генерации звукового сигнала и вызывает эту процедуру с помощью межсегментной команды call:

```
include sysp.inc

      .MODEL  tiny
       DOSSEG
      .STACK  100h

       .DATA

msg    DB 13,10,"Работа в области HMA", 13, 10
      DB "Copyright (C)Frolov A.,1991",13,10,13,10
           DB "$"

noHMM     DB 13,10,"Драйвер HIMEM.SYS "
          DB "не установлен", 13, 10, "$"
yesHMM    DB 13,10,"Драйвер HIMEM.SYS установлен, ","$"
errmsg    DB 13,10,"Ошибка с кодом ", "$"
okmsg     DB 13,10,"Успех!!!", "$"
hmareq    DB 13,10,"Запрашиваем область HMA", "$"
hmarel    DB 13,10,"Освобождаем область HMA", "$"
enA20     DB 13,10,"Открываем линию A20", "$"
dsA20     DB 13,10,"Закрываем линию A20", "$"

HMMEntry dd ?

HMAStart dd ?

      .CODE
      .STARTUP

      mov     ah, 9h          ; Выводим заголовок
      mov     dx, OFFSET msg
      int     21h

; Проверяем, установлен ли драйвер HIMEM.SYS

      mov   ax, 4300h
      int   2fh
```

```
        cmp     al, 80h
        je      HMM_installed

; Если не установлен, выводим сообщение и завершаем
; работу программы

        mov     ah, 9h
        mov     dx, OFFSET noHMM
        int     21h
        jmp     terminate

HMM_installed:

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

; Получаем адрес управляющей функции драйвера

        mov     ax, 4310h
        int     2fh
        mov     word ptr cs:[HMMEntry][0], bx
        mov     word ptr cs:[HMMEntry][2], es

; Запрашиваем область HMA

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

        mov     ax,0100h
        mov     dx,0ffffh
        call    cs:[HMMEntry]
        or      ax, ax
        jnz     hmareq_ok
        jmp     error

hmareq_ok:
        mov     ah, 9h
        mov     dx, OFFSET okmsg
        int     21h

; Открываем линию A20

        mov     ah, 9h
        mov     dx, OFFSET enA20
        int     21h

        mov     ax,0300h
        call    cs:[HMMEntry]
        or      ax, ax
        jnz     enA20_ok
        jmp     error

enA20_ok:
        mov     ah, 9h
```

```
        mov     dx, OFFSET okmsg
        int     21h

; Записываем в двойное слово HMAStart адрес начала области HMA

        mov     word ptr cs:[HMAStart][0], 0010h
        mov     word ptr cs:[HMAStart][2], 0ffffh

; Копируем в область HMA процедуру, которая
; будет там выполняться

        cld
        lea     si, begin_HMA_code
        mov     di, 0010h

        mov     ax, cs
        mov     ds, ax

        mov     ax, 0ffffh
        mov     es, ax

        mov     cx, HMA_code_size
        rep movsb

; Вызываем процедуру, находящуюся в области HMA

        call    cs:[HMAStart]

; Закрываем линию A20

        mov     ah, 9h
        mov     dx, OFFSET dsA20
        int     21h

        mov     ax,0400h
        call    cs:[HMMEntry]
        or      ax, ax
        jnz     dsA20_ok
        jmp     error

dsA20_ok:

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

; Освобождаем область HMA

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

        mov     ax,0200h
        call    cs:[HMMEntry]
        or      ax, ax
        jz      error

        mov     ah, 9h
        mov     dx, OFFSET okmsg
```

```
        int     21h
        jmp     terminate
error:
        push    bx
        mov     ah, 9h
        mov     dx, OFFSET errmsg
        int     21h
        pop     ax
        call    Print_word
terminate:
        .EXIT   0
; Вывод на экран содержимого регистра AX
Print_word proc near
;--------------------
        push ax
        push bx
        push dx
;
        push ax
        mov cl,8
        rol ax,cl
        call Byte_to_hex
        mov bx,dx
        @@out_ch bh
        @@out_ch bl
;
        pop ax
        call Byte_to_hex
        mov bx,dx
        @@out_ch bh
        @@out_ch bl
;
        pop dx
        pop bx
        pop ax
        ret
Print_word endp
;
Byte_to_hex proc near
;--------------------
; al - input byte
; dx - output hex
;--------------------
        push ds
        push cx
        push bx
```

```
        lea bx,tabl
        mov dx,cs
        mov ds,dx
;
        push ax
        and al,0fh
        xlat
        mov dl,al

        pop ax
        mov cl,4
        shr al,cl
        xlat
        mov dh,al
;
        pop bx
        pop cx
        pop ds
        ret
;
tabl db '0123456789ABCDEF'
Byte_to_hex endp

; Эта процедура предназначена для выполнения в области HMA.
; Она просто три раза генерирует звуковой сигнал.

begin_HMA_code:

     BEEP
     BEEP
     BEEP
     retf

end_HMA_code:

; Здесь записана длина процедуры, предназначенной для выполнения
; в области HMA

HMA_code_size dw $-begin_HMA_code
            END
```

В программе имеется макрокоманда BEEP, описанная в файле sysp.inc:

```
; Макро для выдачи звукового сигнала

BEEP  MACRO
      mov bx,0
      mov ax, 0E07h
      int 10h
      ENDM
```

Следующая программа демонстрирует использование функции копирования. Сообщение копируется из области основной памяти в область расширенной памяти, а затем обратно в область основной памяти, но в другое место:

```
include sysp.inc

ExtMemMoveStruct  struc
 @Length      dd ? ; количество пересылаемых байтов
 SourceHandle dw ? ; индекс исходного блока
 SourceOffset dd ? ; смещение в исходном блоке
 DestHandle   dw ? ; индекс блока-назначения
 DestOffset   dd ? ; смещение в блоке
ExtMemMoveStruct  ends

     .MODEL  tiny
     DOSSEG
     .STACK  100h
     .DATA
movestr ExtMemMoveStruct <0,0,0,0,0>
msg DB 13,10,"Использование блоков EMB", 13, 10
      DB  "Copyright (C)Frolov A.,1991",13,10,13,10
      DB "$"
noHMM    DB 13,10,"Драйвер HIMEM.SYS не установлен",13,10,"$"
yesHMM   DB 13,10,"Драйвер HIMEM.SYS установлен, ","$"
errmsg   DB 13,10,"Ошибка с кодом ", "$"
okmsg    DB 13,10,"Успех!!!", "$"
free_ext_mem DB 13,10,"Всего расширенной памяти, Кбайт: ","$"
max_ext_block DB 13,10,"Максимальный участок свободной"
                    DB " расширенной памяти, Кбайт: ", "$"
getEMBmsg  DB 13,10,"Получаем блок EMB","$"
freeEMBmsg DB 13,10,"Освобождаем блок EMB","$"
copymsg    DB 13,10,"Копируем блок данных в область EMB","$"
copymsg1   DB 13,10,"Копируем блок данных обратно","$"
testmsg DB 13,10,13,10,"Сообщение для копирования"
           DB " в область EMB","$"
len_testmsg DW $-testmsg
; Буфер для копирования сообщения
testbuf DB 512 dup(?)
HMMEntry dd ?
EMBHandle dw ?
     .CODE
```

```
    .STARTUP
    mov     ah, 9h                  ; Выводим заголовок
    mov     dx, OFFSET msg
    int     21h
; Проверяем, установлен ли драйвер HIMEM.SYS
    mov   ax, 4300h
    int   2fh
    cmp   al, 80h
    je    HMM_installed
; Если не установлен, выводим сообщение и завершаем
; работу программы
    mov   ah, 9h
    mov   dx, OFFSET noHMM
    int   21h
    jmp   terminate
HMM_installed:
    mov   ah, 9h
    mov   dx, OFFSET yesHMM
    int   21h
; Получаем адрес управляющей функции драйвера
    mov   ax, 4310h
    int   2fh
    mov   word ptr cs:[HMMEntry][0], bx
    mov   word ptr cs:[HMMEntry][2], es
; Определяем размер свободной расширенной памяти
    mov   ah, 9h
    mov   dx, OFFSET free_ext_mem
    int   21h
    mov   ax,0800h
    call  cs:[HMMEntry]
    push  ax
    mov   ax, dx
    call  Print_word
    mov   ah, 9h
    mov   dx, OFFSET max_ext_block
    int   21h
    pop   ax
    call  Print_word
; Получаем блок EMB
    mov   ah, 9h
    mov   dx, OFFSET getEMBmsg
```

```
        int     21h

        mov     ax,0900h
        mov     dx,1h

        call    cs:[HMMEntry]
        or      ax, ax
        jnz     getemb_ok
        jmp     error

getemb_ok:

        mov     EMBHandle, dx

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

; Копируем строку testmsg в блок EMB

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

; Заполняем управляющую структуру

; Длина копируемого массива памяти

        mov     ax, word ptr len_testmsg
        mov     word ptr movestr.@Length, ax

; Индекс основной памяти должен быть = 0

        mov     ax, 0
        mov     word ptr movestr.SourceHandle, ax

; Задаем сегмент:смещение копируемого сообщения

        mov     ax, OFFSET testmsg
        mov     word ptr [movestr.SourceOffset][0], ax
        mov     ax, cs
        mov     word ptr [movestr.SourceOffset][2], ax

; Задаем индекс EMB, в который будем копировать
; сообщение из основной памяти

        mov     ax, EMBHandle
        mov     movestr.DestHandle, ax

; Копируем в начало EMB, поэтому смещение = 0

        mov     ax, 0
        mov     word ptr [movestr.DestOffset][0], ax
        mov     word ptr [movestr.DestOffset][2], ax

; Загружаем адрес управляющей структуры в DS:SI

        mov     ax, cs
        mov     ds, ax
        mov     ax, OFFSET movestr
```

```
        mov     si, ax
        mov     ax,0B00h

; Вызываем функцию копирования

        call    cs:[HMMEntry]
        or      ax, ax
        jnz     moveemb_ok
        jmp     error

moveemb_ok:

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

; Копируем сообщение обратно из блока EMB в буфер testbuf,
; расположенный в основной памяти

        mov     ah, 9h
        mov     dx, OFFSET copymsg1
        int     21h

; Подготавливаем управляющую структуру

        mov     ax, word ptr len_testmsg
        mov     word ptr movestr.@Length, ax

        mov     ax, 0
        mov     word ptr movestr.DestHandle, ax

        mov     ax, OFFSET testbuf
        mov     word ptr [movestr.DestOffset][0], ax
        mov     ax, cs
        mov     word ptr [movestr.DestOffset][2], ax

        mov     ax, EMBHandle
        mov     movestr.SourceHandle, ax

        mov     ax, 0
        mov     word ptr [movestr.SourceOffset][0], ax
        mov     word ptr [movestr.SourceOffset][2], ax

; Выполняем копирование

        mov     ax, cs
        mov     ds, ax
        mov     ax, OFFSET movestr
        mov     si, ax
        mov     ax,0B00h

        call    cs:[HMMEntry]
        or      ax, ax
        jnz     move1emb_ok
        jmp     error

move1emb_ok:
```

```
        mov     ah, 9h
        mov     dx, OFFSET okmsg
        int     21h

; Выводим скопированное сообщение на экран для проверки

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

; Освобождаем блок EMB

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

        mov     ax,0A00h
        mov     dx,EMBHandle

        call    cs:[HMMEntry]
        or      ax, ax
        jnz     freeemb_ok
        jmp     error

freeemb_ok:

        mov     EMBHandle, dx

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

        jmp     terminate

error:
        push    bx

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

        pop     ax
        call    Print_word

terminate:

        .EXIT   0

; Вывод на экран содержимого регистра AX

Print_word proc near
;--------------------
        push ax
        push bx
        push dx
;
        push ax
        mov cl,8
```

```
        rol ax,cl
        call Byte_to_hex
        mov bx,dx
        @@out_ch bh
        @@out_ch bl
;
        pop ax
        call Byte_to_hex
        mov bx,dx
        @@out_ch bh
        @@out_ch bl
;
        pop dx
        pop bx
        pop ax
        ret
Print_word endp
;
Byte_to_hex proc near
;--------------------
; al - input byte
; dx - output hex
;--------------------
        push ds
        push cx
        push bx
;
        lea bx,tabl
        mov dx,cs
        mov ds,dx
;
        push ax
        and al,0fh
        xlat
        mov dl,al
;
        pop ax
        mov cl,4
        shr al,cl
        xlat
        mov dh,al
;
        pop bx
        pop cx
        pop ds
        ret
;
tabl db '0123456789ABCDEF'
Byte_to_hex endp
            END
```

10.6. Интерфейс с Си

Приведем текст программы, позволяющей программам, составленным на языке программирования Си, использовать функции райвера расширенной памяти. Эта программа будет работать только в моделях памяти Small и Compact. Для других моделей памяти требуется изменить строки программы, в которых передаваемые функциям параметры извлекаются из стека и тип процедур (FAR):

Аргументы	Small, Compact	Large, Huge
Первый аргумент	[bp+4]	[bp+6]
Второй аргумент	[bp+6]	[bp+8]

```
; Это интерфейсный модуль для вызова функций XMS из Си.
; Текст программы рассчитан на модель памяти Small.

        .model small,c
        .DATA

; В этом месте будет храниться адрес управляющей функции XMM

XMM_Control  dd   ?

        .CODE

; Макроопределения для выполнения соглашения об
; использовании регистров в процедурах Си

c_begin macro
                push bp
                mov  bp,sp
                push si
                push di
        endm

c_end   macro
                pop  di
                pop  si
                mov  sp,bp
                pop  bp
                ret
        endm

; Все процедуры должны быть public

     public XMM_Installed
     public XMM_Version
     public XMM_RequestHMA
     public XMM_ReleaseHMA
     public XMM_GlobalEnableA20
```

```
        public XMM_GlobalDisableA20
        public XMM_EnableA20
        public XMM_DisableA20
        public XMM_QueryA20
        public XMM_QueryLargestFree
        public XMM_QueryTotalFree
        public XMM_AllocateExtended
        public XMM_FreeExtended
        public XMM_MoveExtended
        public XMM_LockExtended
        public XMM_UnLockExtended
        public XMM_GetHandleLength
        public XMM_GetHandleInfo
        public XMM_ReallocateExtended
        public XMM_RequestUMB
        public XMM_ReleaseUMB
;**
;.Name          XMM_Installed
;.Title         Получение адреса управляющей функции
;
;.Descr         Эта функция проверяет наличие драйвера
;               HIMEM.SYS и в случае его присутствия
;               запоминает адрес управляющей функции.
;
;.Proto         unsigned XMM_Installed(void);
;
;.Params        Не используются
;
;.Return        0 - драйвер HIMEM.SYS не установлен;
;               1 - драйвер HIMEM.SYS установлен.
;
;.Sample        xms_test.c
;**
XMM_Installed proc near
          c_begin

          mov   ax, 4300h
          int   2fh
          cmp   al, 80h
          jne   NotInstalled
          mov   ax, 4310h
          int   2fh
          mov   word ptr [XMM_Control], bx
          mov   word ptr [XMM_Control+2], es
          mov   ax,1
          jmp   Installed

NotInstalled:
          mov   ax, 0
```

```
Installed:
          c_end
XMM_Installed endp

;**
;.Name          XMM_Version
;.Title         Определение версии драйвера HIMEM.SYS
;
;.Descr         Эта функция определяет версию драйвера HIMEM.SYS
;
;.Proto         long  XMM_Version(void);
;
;.Params        Не используются
;
;.Return        Номер версии в младших 16 битах,
;               номер изменений в старших 16 битах
;               возвращаемого значения
;
;.Sample        xms_test.c
;**
XMM_Version proc near
          push si
          push di
          xor  ah, ah
          call [XMM_Control]
          mov  dx, bx
          pop  di
          pop  si
          ret
XMM_Version endp
;**
;.Name          XMM_RequestHMA
;.Title         Запросить область HMA
;
;.Descr         Эта функция пытается зарезервировать для
;               программы область HMA
;
;.Proto         long  XMM_RequestHMA(unsigned space);
;
;.Params        space - размер требуемой области для
;                       TSR-программы или драйвера,
;                       0xffff для прикладной программы;
;
;.Return        < 0 - область HMA не назначена программе,
;                     код ошибки находится в старшем байте.
;               0L  - область HMA назначена программе.
;
;.Sample        xms_test.c
;**
```

```
XMM_RequestHMA proc near
          c_begin
          mov  ah, 1
          mov  dx, [bp+4]
          call [XMM_Control]
          xor  dx, dx
          dec  ax
          jz   @success
          mov  dh, bl
@success:
          c_end
XMM_RequestHMA endp

;**
;.Name          XMM_ReleaseHMA
;.Title         Освободить область HMA
;
;.Descr         Эта функция пытается освободить область HMA
;
;.Proto         long  XMM_ReleaseHMA(void);
;
;.Params        Не используются
;
;.Return        < 0 - область HMA не освобождена,
;                     код ошибки находится в старшем байте.
;               0L - область HMA освобождена.
;
;.Sample        xms_test.c
;**
XMM_ReleaseHMA proc near
          c_begin
          mov  ah, 2
          call [XMM_Control]
          xor  dx, dx
          dec  ax
          jz   @success1
          mov  dh, bl
@success1:
          c_end
XMM_ReleaseHMA endp
;**
;.Name          XMM_GlobalEnableA20
;.Title         Глобальное разрешение линии A20
;
;.Descr         Эта функция разрешает программе, получившей
;               доступ к области HMA, использовать линию A20
;
;.Proto         long  XMM_GlobalEnableA20(void);
;
```

```
;.Params        Не используются
;
;.Return        < 0 - линия A20 не включена,
;                     код ошибки находится в старшем байте.
;               0L  - линия A20 включена.
;
;.Sample        xms_test.c
;**
XMM_GlobalEnableA20 proc near
          c_begin
          mov  ah, 3
          call [XMM_Control]
          xor  dx, dx
          dec  ax
          jz   @success2
          mov  dh, bl
@success2:
          c_end
XMM_GlobalEnableA20 endp
;**
;.Name          XMM_GlobalDisableA20
;.Title         Глобальное запрещение линии A20
;
;.Descr         Эта функция запрещает программе, получившей
;               доступ к области HMA, использовать линию A20
;
;.Proto         long  XMM_GlobalDisableA20(void);
;
;.Params        Не используются
;
;.Return        < 0 - линия A20 не выключена,
;                     код ошибки находится в старшем байте.
;               0L  - линия A20 выключена.
;
;.Sample        xms_test.c
;**
XMM_GlobalDisableA20 proc near
          c_begin
          mov  ah, 4
          call [XMM_Control]
          xor  dx, dx
          dec  ax
          jz   @success3
          mov  dh, bl
@success3:
          c_end
XMM_GlobalDisableA20 endp
```

```
;**
;.Name          XMM_EnableA20
;.Title         Локальное разрешение линии A20
;
;.Descr         Эта функция разрешает программе управлять
;               областью расширенной памяти.
;
;.Proto         long  XMM_EnableA20(void);
;
;.Params        Не используются
;
;.Return        < 0 - линия A20 не включена,
;                     код ошибки находится в старшем байте.
;               0L  - линия A20 включена.
;
;.Sample        xms_test.c
;**
XMM_EnableA20 proc near
          c_begin
          mov  ah, 5
          call [XMM_Control]
          xor  dx, dx
          dec  ax
          jz   @success4
          mov  dh, bl
@success4:
          c_end
XMM_EnableA20 endp
;**
;.Name          XMM_DisableA20
;.Title         Локальное запрещение линии A20
;
;.Descr         Эта функция запрещает программе управлять
;               областью расширенной памяти.
;
;.Proto         long  XMM_DisableA20(void);
;
;.Params        Не используются
;
;.Return        < 0 - линия A20 не выключена,
;                     код ошибки находится в старшем байте.
;               0L  - линия A20 выключена.
;
;.Sample        xms_test.c
;**
XMM_DisableA20 proc near
          c_begin
          mov  ah, 6
          call [XMM_Control]
```

```
          xor   dx, dx
          dec   ax
          jz    @success5
          mov   dh, bl
@success5:
          c_end
XMM_DisableA20 endp

;**
;.Name          XMM_QueryA20
;.Title         Проверить состояние линии A20
;
;.Descr         Эта функция проверяет доступность линии A20
;
;.Proto         long  XMM_QueryA20(void);
;
;.Params        Не используются
;
;.Return        < 0 - ошибка,
;                     код ошибки находится в старшем байте.
;               0L  - линия A20 выключена,
;               1L  - линия A20 включена.
;
;.Sample        xms_test.c
;**
XMM_QueryA20 proc near
          c_begin
          mov   ah, 7
          call [XMM_Control]
          xor   dx, dx
          or    ax, ax
          jnz   @success6
          mov   dh, bl
@success6:
          c_end
XMM_QueryA20 endp

;**
;.Name          XMM_QueryLargestFree
;.Title         Определить максимальный размер блока
;
;.Descr         Эта функция возвращает размер максимального
;               непрерывного блока расширенной памяти,
;               который доступен программе.
;
;.Proto         long  XMM_QueryLargestFree(void);
;
;.Params        Не используются
;
;.Return        < 0 - ошибка,
;                     код ошибки находится в старшем байте.
```

```
;               >= 0 - размер блока.
;
;.Sample        xms_test.c
;**
XMM_QueryLargestFree proc near
          c_begin
          mov  ah, 8
          call [XMM_Control]
          xor  dx, dx
          or   ax, ax
          jnz  @success7
          mov  dh, bl
@success7:
          c_end
XMM_QueryLargestFree endp
;**
;.Name          XMM_QueryTotalFree
;.Title         Определить размер расширенной памяти
;
;.Descr         Эта функция возвращает размер
;               всей имеющейся расширенной памяти.
;
;.Proto         long  XMM_QueryTotalFree(void);
;
;.Params        Не используются
;
;.Return        < 0 - ошибка,
;                     код ошибки находится в старшем байте.
;               >= 0 - размер расширенной памяти.
;
;.Sample        xms_test.c
;**
XMM_QueryTotalFree proc near
          c_begin
          mov  ah, 8
          call [XMM_Control]
          or   ax, ax
          mov  ax, dx
          mov  dx, 0
          jnz  @success8
          mov  dh, bl
@success8:
          c_end
XMM_QueryTotalFree endp
;**
;.Name          XMM_AllocateExtended
;.Title         Запросить блок расширенной памяти
;
```

```
;.Descr         Эта функция выделяет программе блок
;               расширенной памяти, в случае успеха
;               возвращает индекс полученного блока.
;
;.Proto         long XMM_AllocateExtended(unsigned space);
;
;.Params        space - размер требуемого блока памяти
;                       в килобайтах;
;
;.Return        < 0 - блок не распределен,
;                     код ошибки находится в старшем байте.
;               > 0L  - младший байт содержит индекс
;                       полученного блока памяти.
;
;.Sample        xms_test.c
;**

XMM_AllocateExtended proc near
          c_begin
          mov  ah, 9
          mov  dx,  [bp+4]
          call [XMM_Control]
          or   ax, ax
          mov  ax, dx
          mov  dx, 0
          jnz  @success9
          mov  dh, bl
@success9:
          c_end
XMM_AllocateExtended endp
;**
;.Name          XMM_FreeExtended
;.Title         Освободить блок расширенной памяти
;
;.Descr         Эта функция освобождает блок
;               расширенной памяти, полученный функцией
;               XMM_AllocateExtended().
;
;.Proto         long XMM_FreeExtended(unsigned handle);
;
;.Params        handle - индекс освобождаемого блока памяти;
;
;.Return        < 0 - блок не распределен,
;                     код ошибки находится в старшем байте.
;               0L  - блок освобожден.
;
;.Sample        xms_test.c
;**
```

```
XMM_FreeExtended proc near
          c_begin
          mov  ah, 0Ah
          mov  dx, [bp+4]
          call [XMM_Control]
          xor  dx, dx
          dec  ax
          jz   @successA
          mov  dh, bl
@successA:
          c_end
XMM_FreeExtended endp

;**
;.Name          XMM_MoveExtended
;.Title         Копировать блок расширенной памяти
;
;.Descr         Эта функция копирует блок расширенной памяти,
;               используя структуру struct XMM_Move:
;
;                  struct   XMM_Move {
;                     unsigned long  Length;
;                     unsigned short SourceHandle;
;                     unsigned long  SourceOffset;
;                     unsigned short DestHandle;
;                     unsigned long  DestOffset;
;                  };
;
;.Proto         long  XMM_MoveExtended(struct
;                        XMM_Move *move_descr);
;
;.Params        struct XMM_Move *move_descr -
;                  указатель на структуру, описывающую,
;                  что, откуда и куда надо копировать.
;
;.Return        < 0 - ошибка при копировании,
;                     код ошибки находится в старшем байте.
;               0L  - блок скопирован успешно.
;
;.Sample        xms_test.c
;**
XMM_MoveExtended proc near
          c_begin
          mov  ah, 0Bh
          mov  si, [bp+4];
          call [XMM_Control]
          xor  dx, dx
          dec  ax
          jz   @successB
          mov  dh, bl
```

```
@successB:
          c_end
XMM_MoveExtended endp

;**
;.Name          XMM_LockExtended
;.Title         Заблокировать блок расширенной памяти
;
;.Descr         Эта функция блокирует блок расширенной памяти и
;               возвращает 31 разряд его физического адреса.
;
;.Proto         long XMM_LockExtended(unsigned handle);
;
;.Params        handle - индекс блокируемого блока памяти;
;
;.Return        < 0 - блок не заблокирован,
;                     код ошибки находится в старшем байте.
;               > 0L  - блок заблокирован, функция возвращает
;                       физический адрес блока памяти.
;
;.Sample        xms_test.c
;**
XMM_LockExtended proc near
          c_begin
          mov  ah, 0Ch
          mov  dx, [bp+4]
          call [XMM_Control]
          xchg ax, bx
          dec  bx
          jz   XMML_Success
          mov  dh, al
XMML_Success:
          c_end
XMM_LockExtended endp

;**
;.Name          XMM_UnLockExtended
;.Title         Разблокировать блок расширенной памяти
;
;.Descr         Эта функция разблокирует блок расширенной памяти.
;
;.Proto         long XMM_UnLockExtended(unsigned handle);
;
;.Params        handle - индекс блока памяти;
;
;.Return        < 0 - блок не разблокирован,
;                     код ошибки находится в старшем байте.
;               0L  - блок разблокирован.
;
;.Sample        xms_test.c
;**
```

```
XMM_UnLockExtended proc near
          c_begin
          mov  ah, 0Dh
          mov  dx, [bp+4]
          call [XMM_Control]
          xor  dx, dx
          dec  ax
          jz   @successC
          mov  dh, bl
@successC:
          c_end
XMM_UnLockExtended endp

;**
;.Name          XMM_GetHandleLength
;.Title         Получить длину блока расширенной памяти
;
;.Descr         Эта функция возвращает длину блока
;               расширенной памяти по его индексу.
;
;.Proto         long XMM_GetHandleLength(unsigned handle);
;
;.Params        handle - индекс блока памяти;
;
;.Return        < 0 - произошла ошибка,
;                     код ошибки находится в старшем байте.
;               > 0L  - длина блока в килобайтах.
;
;.Sample        xms_test.c
;**
XMM_GetHandleLength proc near
          c_begin
          mov  ah, 0Eh
          mov  dx, [bp+4]
          call [XMM_Control]
          or   ax, ax
          mov  ax, dx
          mov  dx, 0
          jnz  @successD
          mov  dh, bl
@successD:
          c_end
XMM_GetHandleLength endp

;**
;.Name          XMM_GetHandleInfo
;.Title         Получить информацию о блоке расширенной памяти
;
;.Descr         Эта функция возвращает общее количество индексов
;               в системе и содержимое счетчика блокирования для
;               заданного индекса.
```

```
;.Proto         long XMM_GetHandleInfo(unsigned handle);
;
;.Params        handle - индекс блока памяти;
;
;.Return        < 0 - произошла ошибка,
;                     код ошибки находится в старшем байте.
;               > 0L  - младший байт - общее количество
;                       индексов в системе;
;                       старший байт - счетчик блокирования.
;
;.Sample        xms_test.c
;**
XMM_GetHandleInfo proc near
          c_begin
          mov  ah, 0Eh
          mov  dx, [bp+4]
          call [XMM_Control]
          mov  dx, bx
          or   ax, ax
          mov  ax, dx
          mov  dx, 0
          jnz  @successE
          mov  dh, bl
@successE:
          c_end
XMM_GetHandleInfo endp
;**
;.Name          XMM_ReallocateExtended
;.Title         Изменить размер блока расширенной памяти
;
;.Descr         Эта функция изменяет размер выделенного
;               блока расширенной памяти.
;
;.Proto         long XMM_ReallocateExtended(unsigned handle,
;                  unsigned new_size);
;
;.Params        handle - индекс блока памяти;
;               new_size - новый размер блока памяти
;                       в килобайтах;
;
;.Return        < 0 - блок не распределен,
;                     код ошибки находится в старщем байте.
;               > 0L  - младший байт содержит индекс
;                       полученного блока памяти.
;
;.Sample        xms_test.c
;**
XMM_ReallocateExtended proc near
          c_begin
```

```
          mov  ah, 0Fh
          mov  dx, [bp+4]
          mov  bx, [bp+6]
          call [XMM_Control]
          xor  dx, dx
          dec  ax
          jz   @successF
          mov  dh, bl
@successF:
          c_end
XMM_ReallocateExtended endp

;**
;.Name          XMM_RequestUMB
;.Title         Запросить область UMB
;
;.Descr         Эта функция пытается зарезервировать для
;               программы область UMB
;
;.Proto         long  XMM_RequestUMB(unsigned space);
;
;.Params        space - размер требуемой области
;                       в параграфах;
;
;.Return        < 0 - область UMB не назначена программе,
;                     код ошибки находится в старшем байте;
;                     максимальный размер доступного блока
;                     в младшем слове (16 разрядов);
;               > 0L  - область UMB назначена программе,
;                     младшее слово содержит сегмент блока UMB,
;                     старший - размер выделенного блока UMB.
;
;.Sample        xms_test.c
;**
XMM_RequestUMB proc near
          c_begin
          mov  ah, 10h
          mov  dx, [bp+4]
          call [XMM_Control]
          xchg bx, ax
          dec  bx
          jz   RUMB_Success
          xchg ax, dx
          mov  dh, dl
RUMB_Success:
          c_end
XMM_RequestUMB endp

;**
;.Name          XMM_ReleaseUMB
;.Title         Освободить область UMB
```

```
;.Descr         Эта функция пытается освободить область UMB
;
;.Proto         long  XMM_ReleaseUMB(unsigned segment);
;
;.Params        segment - сегмент освобождаемого блока UMB*
;
;.Return        < 0 - область UMB не освобождена,
;                     код ошибки находится в старшем байте.
;               0L - область UMB освобождена.
;
;.Sample        xms_test.c
;**
XMM_ReleaseUMB proc near
          c_begin
          mov  ah, 11h
          mov  dx, [bp+4]
          call [XMM_Control]
          xor  dx, dx
          dec  ax
          jz   @success10
          mov  dh, bl
@success10:
          c_end
XMM_ReleaseUMB endp
           END
```

Приведем пример программы, демонстрирующей использование некоторых функций XMM:

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

void main(void);
void main(void) {

     long ver, rc, handle;
     static char testmsg[] = "Тестовое сообщение";
     char buf[80];
     char far *ptr;
     int i;
     struct XMM_Move move_d;

// Проверяем, установлен ли драйвер HIMEM.SYS,
// если установлен, выводим его версию.

     if (XMM_Installed()) {
          printf("\nДрайвер HIMEM.SYS установлен.");
          ver = XMM_Version();
          printf("\nВерсия XMM: %4X, изменения: %4x",
```

```
                (short)ver, (short)(ver >> 16));
     }

     else {
          printf("\nДрайвер HIMEM.SYS не установлен.");
          exit(-1);
     }

// Запрашиваем управление областью HMA.

     rc = XMM_RequestHMA(0xffff);
     if(rc) error("Ошибка при запросе области HMA",rc);

     else {

// Открываем линию A20.

          rc = XMM_GlobalEnableA20();
          if(rc) error("Ошибка при разрешении линии A20",rc);

// Копируем тестовое сообщение сначала из
// стандартной памяти в область HMA,
// затем обратно в стандартную память.

          ptr = FP_MAKE(0xffff,0x0010);

          for(i=0; testmsg[i] != 0; i++)
               ptr[i] = testmsg[i];

          for(i=0; ptr[i] != 0; i++)
               buf[i] = ptr[i];
          buf[i] = 0;

// Выводим сообщение для проверки.

          printf("\n%s",buf);

// Закрываем линию A20 и отдаем системе область HMA.

          rc = XMM_GlobalDisableA20();
          if(rc) error("Ошибка при запрещении линии A20",rc);

          rc = XMM_ReleaseHMA();
          if(rc) error("Ошибка при освобождении области
HMA",rc);
     }

// Получаем блок EMB размером в 1 килобайт.

     handle = XMM_AllocateExtended(1);
     if(handle < 0) error("Ошибка при запросе XMB",handle);

// Копируем тестовое сообщение сначала из
// стандартной памяти в блок EMB,
// затем обратно в стандартную память.
```

```
    move_d.Length = strlen(testmsg) + 1;
    move_d.SourceHandle = 0;
    (char far*)move_d.SourceOffset = (char far*)testmsg;
    move_d.DestHandle = handle;
    move_d.DestOffset = 0L;

    rc = XMM_MoveExtended(&move_d);
    if(rc < 0) error("Ошибка при копировании в EMB",rc);

    move_d.Length = strlen(testmsg) + 1;
    move_d.DestHandle = 0;
    (char far*)move_d.DestOffset = (char far*)buf;
    move_d.SourceHandle = handle;
    move_d.SourceOffset = 0L;

    rc = XMM_MoveExtended(&move_d);
    if(rc < 0) error("Ошибка при копировании из EMB",rc);

// Выводим сообщение для проверки.

    printf("\n%s",buf);

// Освобождаем блок EMB.

    rc = XMM_FreeExtended(handle);
    if(rc) error("Ошибка при освобождении XMB",rc);

    exit(0);
}

// Функция для вывода сообщения об ошибке
// и кода ошибки.

int error(char *msg, long rc) {

    rc = (unsigned char)(rc >> 24) ;
    printf("\n%s, код ошибки: %02.2X\n",
        msg, (unsigned char)rc);
}
```

Глава 11

ДОПОЛНИТЕЛЬНАЯ ПАМЯТЬ

В отличие от расширенной памяти дополнительная память с помощью специальной аппаратуры и программного обеспечения отображается в диапазон адресов, лежащий ниже границы 1 мегабайт. Такой способ пригоден для компьютеров, использующих процессор Intel 8086, не обладающий возможностью адресации расширенной памяти.

Чтобы понять, как происходит отображение, вспомним распределение первого мегабайта оперативной памяти. Область с адресами от 00000 до 9FFFF - это стандартная память размером 640 килобайт. (Мы используем здесь физический 20-разрядный адрес.) Диапазон адресов от A0000 до BFFFF используется видеоадаптерами. Наконец, 256 килобайт с адресами C0000 - FFFFF используется для BIOS.

Однако обычно BIOS не занимает все 256 килобайт адресного пространства, оставляя окна размером в десятки килобайтов. С помощью специальной аппаратуры можно отображать эти окна на участки дополнительной памяти, как бы "подставляя" участки памяти под адреса окон.

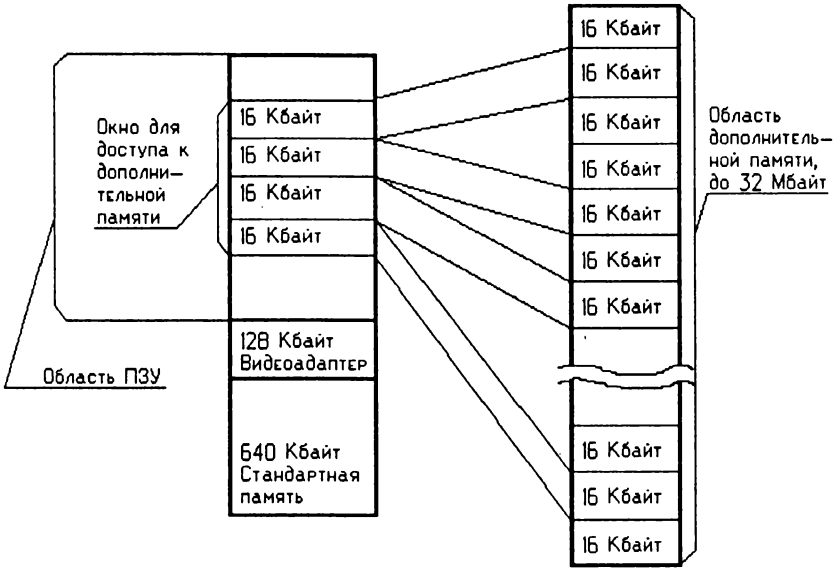
В практике построения вычислительных систем на основе микропроцессоров такая техника используется уже давно. Для компьютеров IBM PC/XT/AT корпорации Lotus Development, Intel и Microsoft разработали спецификацию расширенной памяти (Expanded Memory Specification - EMS). В настоящий момент распространены версии 3.2 и 4.0 этой спецификации.

Существует альтернативная спецификация Extended Expanded Memory Specification - EEMS), отличающаяся от версии EMS 3.2 в основном наличием поддержки мультизадачных операционных систем.

Эта спецификация была разработана другой группой крупных производителей компьютерного оборудования: AST Research, Ashton-Tate, Quadram. Однако эта спецификация не получила широкого распространения. Версия 4.0 EMS включила в себя все расширения спецификации EEMS.

В спецификации EMS в качестве окна для доступа к дополнительной памяти используются 64 килобайта, расположенные по адресам C0000h - EFFFFh. Это окно в спецификации называется page frame. Окно разбивается на четыре сегмента по 16 килобайтов. Вся дополнительная память разбивается на логические страницы (logical page) размером по 16 килобайтов. Любая логическая страница может быть отображена на любой сегмент окна доступа. Таким образом, используя четыре сегмента, программа может адресоваться одновременно к любым четырем логическим страницам дополнительной памяти.

На рисунке схематически показано отображение логических страниц дополнительной памяти на сегменты 64-килобайтного окна, расположенного в области адресов ПЗУ:



11.1. Драйверы дополнительной памяти

Для использования дополнительной памяти в компьютер должна быть вставлена плата дополнительной памяти и в файле CONFIG.SYS подключен специальный драйвер, который обычно

поставляется вместе с платой памяти. Драйвер выполняет управление дополнительной памятью и называется EMM (Expanded Memory Manager).

Операционная система MS-DOS версии 4.01 содержит драйвер XMA2EMS.SYS, реализующий функции управления дополнительной памятью. Этот драйвер должен быть подключен в файле CONFIG.SYS следующим образом:

```
DEVICE=XMA2EMS.SYS [FRAME=xxxx] [Pnnn=xxxx] [/X:pages]
```

Параметр FRAME задает базовый адрес для 64-килобайтового окна доступа в виде шестнадцатеричного сегментного адреса, например C000. Адрес должен находиться в диапазоне C000 - E000.

Параметр Pnnn позволяет задать базовый адрес для конкретной страницы дополнительной памяти. Здесь nnn - это номер страницы (0-255), xxxx - сегментный адрес в шестнадцатеричном формате. При использовании параметра FRAME нельзя указывать параметры P0, P1, P2, P3.

Параметр /X:pages определяет, сколько страниц дополнительной памяти будет использовано. По умолчанию используется вся дополнительная память.

Если Ваш компьютер имеет процессор 80386 и расширенную память, Вы можете использовать драйвер EMM386.SYS, поставляемый в составе MS-DOS версии 4.01. Этот драйвер эмулирует дополнительную память на расширенной памяти. При этом несколько снижается производительность системы.

Драйвер может быть подключен следующим образом:

```
DEVICE=EMM386.SYS [size]  [X:mmmm-nnnn] [Mx]
```

Параметр size определяет количество используемой драйвером расширенной памяти в килобайтах. Значение по умолчанию - 256 килобайт.

Параметр X:mmmm-nnnn определяет диапазон памяти, которая не должна быть использована для размещения окон доступа.

Параметр Mx задает расположение окна доступа, используемого для отображения логических страниц дополнительной памяти. Соответствие параметра x сегментному адресу окна приведено в таблице:

x	*Адрес окна доступа*	*x*	*Адрес окна доступа*
0	C000	5	D400
1	C400	6	D800
2	C800	7	DC00
3	CC00	8	E000
4	D000		

11.2. Проверка подключения драйвера

Драйвер дополнительной памяти устанавливает вектор прерывания INT 67h таким образом, что этот вектор указывает на заголовок драйвера. При изучении драйверов мы рассказывали Вам о формате заголовка. Сейчас для нас важно, что со смещением 10 в заголовке располагается имя драйвера - "EMMXXXX0". Следовательно, для проверки подключения драйвера мы можем, получив адрес заголовка, сравнить 8 байтов имени устройства со строкой "EMMXXXX0". При совпадении мы можем считать, что драйвер дополнительной памяти установлен.

Для проверки установки драйвера Вы можете использовать следующую функцию, выполняющую все описанные действия:

```
/**
*.Name          ems_init
*.Title         Функция проверяет установку драйвера EMS
*
*.Descr         Эта функция проверяет наличие драйвера EMS
*
*.Proto         int ems_init(void);
*
*.Params        Не используются
*
*.Return        0 - драйвер EMS установлен;
*               1 - драйвер EMS не установлен.
*
*.Sample        ems_test.c
**/

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

int ems_init(void) {
```

```
    void (_interrupt _far *EMS_driver_adr)(void);
    char _far *EMS_driver_name;
    char test_name[8];
    int i;

    EMS_driver_adr = _dos_getvect(0x67);

    FP_SEG(EMS_driver_name) = FP_SEG (EMS_driver_adr);
    FP_OFF(EMS_driver_name) = 10;

    for(i=0; i<8; i++) test_name[i] = EMS_driver_name[i];

    if(strncmp(test_name, "EMMXXXX0", 8) == 0) return(0);
    else return(1);
}
```

11.3. Вызов функций драйвера

Для вызова функций драйвера дополнительной памяти программа должна загрузить код функции в регистр AH, код подфункции (обычно 0) - в регистр AL и затем вызвать прерывание INT 67h.

После возврата из прерывания в регистр AH будет записано слово состояния. При успешном выполнении функции оно равно нулю.

11.4. Стандартные функции EMM

Стандартные функции - это небольшое подмножество функций EMM, необходимое для работы обычных прикладных программ (не резидентных и не драйверов). Все эти функции поддерживаются EMM версии 3.2.

11.4.1. Получить состояние EMM

На входе: AX = 4000h.

На выходе: AH = байт состояния EMM.

Эта функция используется для проверки состояния драйвера EMM. Она должна использоваться только после того, как программа убедилась в наличии драйвера EMM.

Для получения состояния EMM используйте следующую функцию:

```
/**
*.Name          ems_stat
*.Title         Определение состояния драйвера EMS
*
*.Descr         Эта функция возвращает байт состояния
*               драйвера EMS
*
*.Proto         char ems_stat(void);
*
*.Params        Не используются
*
*.Return        Байт состояния драйвера EMS
*
*.Sample        ems_test.c
**/

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

char ems_stat(void) {

     union REGS reg;
     struct SREGS sreg;

     reg.x.ax = 0x4000;
     int86(0x67, &reg, &reg);
     return(reg.h.ah);
}
```

11.4.2. Получить сегмент окна

На входе: AX = 4100h.

На выходе: AH = байт состояния EMM;
BX = сегмент окна для доступа к логическим страницам дополнительной памяти.

Функция позволяет получить сегмент 64-килобайтного окна, используемого драйвером EMS для доступа к логическим страницам расширенной памяти.

```
/**
*.Name          ems_fram
*.Title         Определение сегмента окна доступа
*
*.Descr         Эта функция возвращает сегментный адрес
*               окна, которое используется для доступа к
*               дополнительной памяти.
*
```

```
*.Proto         char ems_fram(unsigned *frame);
*
*.Params        unsigned *frame - Указатель на переменную,
*                  в которую будет записан сегментный
*                  адрес окна доступа.
*
*.Return        Состояние EMM.
*
*.Sample        ems_test.c
**/

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

char ems_fram(unsigned *frame) {

     union REGS reg;
     struct SREGS sreg;

     reg.x.ax = 0x4100;
     int86(0x67, &reg, &reg);
     *frame = reg.x.bx;

     return(reg.h.ah);
}
```

11.4.3. Получить размер доступной памяти EMS

На входе: AX = 4200h.

На выходе: AH = байт состояния EMM;
DX = общее количество 16-килобайтных страниц EMS в системе;
BX = число доступных в настоящее время страниц EMS.

Эта функция позволяет Вам получить информацию о наличии и доступности страниц дополнительной памяти.

```
/**
*.Name          ems_page
*.Title         Определение количества страниц EMS
*
*.Descr         Эта функция предназначена для определения
*               общего количества страниц EMS и количества
*               страниц, доступных в настоящее время.
*
*.Proto         char ems_page(unsigned *total, unsigned *free);
```

```
*
*.Params        unsigned *total - указатель на переменную,
*                  в которую будет записано общее количество
*                  страниц памяти EMS;
*               unsigned *free - указатель на переменную,
*                  в которую будет записано количество
*                  доступных страниц памяти EMS;
*
*.Return        Состояние EMM.
*
*.Sample        ems_test.c
**/

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

char ems_page(unsigned *total, unsigned *free) {

     union REGS reg;

     reg.x.ax = 0x4200;
     int86(0x67, &reg, &reg);
     *total = reg.x.dx;
     *free  = reg.x.bx;

     return(reg.h.ah);
}
```

11.4.4. Открыть индекс EMM

На входе: AX = 4300h;
BX = требуемое в данном пуле количество логических страниц.

На выходе: AH = байт состояния EMM;
DX = индекс пула EMS, он будет использоваться в операциях с пулом логических страниц.

Эта функция позволяет заказать пул логических страниц (т. е. некоторую совокупность логических страниц дополнительной памяти). Полученному пулу присваивается индекс (handle), который указывает на пул и используется во всех операциях с пулом.

```
/**
*.Name          ems_open
*.Title         Открытие индекса пула страниц EMS
```

```
*
*.Descr         Эта функция открывает индекс пула страниц
*               EMS, делая доступными логические страницы
*               дополнительной памяти.
*
*.Proto         int ems_open(int n_pages, int *handle);
*
*.Params        int n_pages - количество требуемых логических
*                  страниц;
*               int *handle - указатель на слово, в которое
*                  будет записан индекс полученного пула.
*
*.Return        Байт состояния драйвера EMS
*
*.Sample        ems_test.c
**/

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

int ems_open(int n_pages, int *handle) {

     union REGS reg;

     reg.x.ax = 0x4300;
     reg.x.bx = n_pages;
     int86(0x67, &reg, &reg);

     *handle = reg.x.dx;
     return(reg.h.ah);
}
```

11.4.5. Отобразить память

На входе: AH = 44h;
AL = номер физической страницы окна доступа (от 0 до 3);
BX = номер логической страницы из числа находящихся в пуле страниц (от 0 до n-1, где n - количество логических страниц в пуле); для версии EMS 4.0 задание значения 0FFFFh приводит к запрещению отображения физических страниц пула, для разрешения их отображения необходимо вызвать эту

			функцию еще раз, указав правильный номер страницы;
	DX	=	индекс EMM, полученный от функции 43h.
На выходе:	AH	=	байт состояния EMM.

Функция выполняет отображение (привязку) одной из логических страниц пула к одному их четырех 16-килобайтных сегментов окна просмотра, т. е. к физическим страницам.

```
/**
*.Name          ems_map
*.Title         Отобразить память EMS
*
*.Descr         Эта функция отображает логические страницы
*               пула дополнительной памяти на физические.
*
*.Proto         int ems_map(int phys_page, int log_page,
*                           int handle);
*
*.Params        int phys_pages - номер физической страницы
*                  окна доступа (от 0 до 3), на которую
необходимо
*                  отобразить логическую страницу пула;
*
*               int_log_page - номер логической страницы пула;
*
*               int *handle - индекс полученного пула;
*
*.Return        Байт состояния драйвера EMS
*
*.Sample        ems_test.c
**/

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

int ems_map(int phys_page, int log_page, int handle) {

     union REGS reg;

     reg.h.ah = 0x44;
     reg.h.al = phys_page;
     reg.x.bx = log_page;
     reg.x.dx = handle;
     int86(0x67, &reg, &reg);

     return(reg.h.ah);
}
```

11.4.6. Закрыть индекс EMM

На входе: AX = 4500h;
DX = индекс EMM.
На выходе: AH = байт состояния EMM.

Функция освобождает все логические страницы пула. После освобождения эти страницы могут быть повторно распределены.

```
/**
*.Name          ems_clos
*.Title         Закрытие индекса пула страниц EMS
*
*.Descr         Эта функция закрывает индекс пула страниц,
*               полученный функцией ems_open().
*
*.Proto         int ems_clos(int *handle);
*
*.Params        int *handle - указатель на слово, в которое
*                  будет записан индекс полученного пула.
*
*.Return        Байт состояния драйвера EMS
*
*.Sample        ems_test.c
**/

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

int ems_clos(int *handle) {

     union REGS reg;

     reg.x.ax = 0x4500;
     reg.x.dx = *handle;
     int86(0x67, &reg, &reg);

     return(reg.h.ah);
}
```

11.4.7. Получить номер версии EMM

На входе: AX = 4600h.
На выходе: AH = байт состояния EMM;
AL = номер версии в двоично-десятичном (BCD) формате, 32h соответствует версии 3.2.

Версия 4.0 EMM поддерживает больше функций по управлению дополнительной памятью, чем версия 3.2. Прежде чем использовать такие функции, следует определить номер используемой версии EMM с помощью функции 46h.

```
/**
*.Name          ems_ver
*.Title         Определение версии драйвера EMS
*
*.Descr         Эта функция возвращает номер версии
*               драйвера EMS в двоично-десятичном формате.
*
*.Proto         int ems_ver(char *ver);
*
*.Params        char *ver - указатель на байт, в который
*                  будет записан номер версии.
*
*.Return        Номер версии драйвера EMS в формате BCD
*
*.Sample        ems_test.c
**/

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

int ems_ver(char *ver) {

     union REGS reg;

     reg.x.ax = 0x4600;
     int86(0x67, &reg, &reg);

     *ver = reg.h.al;
     return(reg.h.ah);
}
```

11.5. Дополнительные функции EMM

Дополнительные функции используются резидентными программами, драйверами и мультизадачными приложениями. Кроме того, с помощью этих функций можно выполнять пересылку массивов в дополнительной и стандартной памяти, а также организовать подкачку и выполнение программных модулей в дополнительной памяти.

Обратите внимание, что использование расширенной памяти в спецификации XMS позволяет выполнять программные модули (с

ограничениями) только в области HMA размером примерно 64 килобайта. Остальная расширенная память может быть использована только для хранения данных.

Специальные функции с номерами 47h - 4Eh поддерживаются EMM версии 3.2, функции с номерами 4Fh - 58h - только EMM версии 4.0.

11.5.1. Сохранить контекст отображения

На входе:	AX	=	4700h;
	DX	=	индекс EMM.
На выходе:	AH	=	байт состояния EMM.

Эта функция предназначена для использования драйверами и резидентными программами. Прежде чем работать с дополнительной памятью, такие программы должны сохранить текущий контекст отображения логических страниц. Перед возвратом управления прикладной программе контекст должен быть восстановлен с помощью функции 48h.

11.5.2. Восстановить контекст отображения

На входе:	AX	=	4800h;
	DX	=	индекс EMM.
На выходе:	AH	=	байт состояния EMM.

Функция позволяет восстановить контекст отображения логических страниц пула, сохраненный предыдущей функцией.

11.5.3. Определить количество страниц в пуле

На входе:	AX	=	4B00h;
	DX	=	индекс EMM.
На выходе:	AH	=	байт состояния EMM;
	BX	=	количество логических страниц в пуле.

Функция возвращает количество логических страниц дополнительной памяти для выбранного пула.

11.5.4. Определить количество активных пулов

На входе: AX = 4C00h;
DX = индекс EMM.
На выходе: AH = байт состояния EMM;
BX = количество активных пулов дополнительной памяти.

Функция предназначена для определения количества текущих активных пулов и может вызываться перед использованием следующей функции (с номером 4Dh).

11.5.5. Получить информацию о пулах

На входе: AX = 4D00h;
ES:DI = адрес буфера для информации.
На выходе: AH = байт состояния EMM;
BX = количество активных пулов дополнительной памяти.

После вызова функции буфер заполняется как массив структур, содержащих два 16-битовых слова. Первое слово содержит индекс пула, второе - количество логических страниц в пуле.

11.5.6. Получить/установить отображение всех страниц

На входе: AH = 4Eh;
AL = код подфункции:
0 - получить содержимое всех регистров отображения в буфер ES:DI;
1 - восстановить содержимое всех регистров отображения из буфера ES:DI;
2 - получить и установить все регистры отображения в буфер ES:DI или восстановить из буфера ES:DI (т. е. комбинация подфункций 0 и 1);
3 - определить размер требуемого буфера;

	ES:DI	=	адрес буфера для информации;
На выходе:	AH	=	байт состояния EMM;
	AL	=	для подфункции 3 - размер буфера.

Эта функция предназначена для поддержки мультизадачности. Она позволяет быстро изменять контекст дополнительной памяти при переключении с одного процесса на другой.

11.5.7. Получить/установить отображение части страниц

На входе:	AH	=	Fh;
	AL	=	код подфункции: 0 - получить содержимое регистров отображения в буфер ES:DI; 1 - восстановить содержимое регистров отображения из буфера ES:DI; 2 - определить размер требуемого буфера;
	ES:DI	=	адрес буфера для информации;
	DS:SI	=	адрес массива 16-битовых слов, содержащих сегментные адреса страниц, для которых необходимо сохранить контекст отображения; самое первое слово массива - это размер массива в словах.
На выходе:	AH	=	байт состояния EMM;
	AL	=	для подфункции 2 - размер буфера.

Функция работает аналогично предыдущей, но позволяет сохранять контексты только для некоторых страниц. Это экономит время, необходимое для сохранения/восстановления.

11.5.8. Отображение/запрещение группы страниц

На входе:	AH	=	50h;
	AL	=	код подфункции: 0 - разрешить или запретить отображение страниц, используя номера страниц;

			1 - аналогично подфункции 0, но используются не номера страниц, а адреса сегментов;
	DS:SI	=	адрес массива структур, состоящий из двух слов, первое слово - номер логической страницы, второе - номер физической страницы; указание логической страницы 0FFFFh запрещает отображение страницы;
	CX	=	размер массива DS:SI в количестве структур.
На выходе:	AH	=	байт состояния EMM.

Функция позволяет за один вызов разрешить и запретить использование нескольких страниц.

11.5.9. Изменение размера пула

На входе:	AX	=	5100h;
	BX	=	новый размер пула в логических страницах;
	DX	=	индекс EMM.
На выходе:	AH	=	байт состояния EMM.

С помощью этой функции программа может изменить размер уже заказанного ранее пула страниц дополнительной памяти.

11.5.10. Получить/установить атрибуты пула

На входе:	AH	=	52h;
	AL	=	код подфункции: 0 - получить атрибуты пула; 1 - установить атрибуты пула; 2 - определить возможность использования атрибута неразрушаемой памяти;
	BL	=	новые атрибуты;
	DX	=	индекс EMM.

На выходе: AH = байт состояния EMM;
AL = для подфункции 0: атрибуты пула;
для подфункции 2:
0 - атрибут неразрушаемой памяти недоступен;
1 - атрибут неразрушаемой памяти доступен.

Функция дает возможность установить для некоторых пулов атрибут неразрушаемой памяти. Содержимое таких пулов не исчезает при теплой перезагрузке операционной системы (после нажатия комбинации клавиш Ctrl-Alt-Del).

11.5.11. Установить/прочитать имя пула

На входе: AH = 53h;
AL = код подфункции:
0 - получить имя пула;
1 - установить имя пула;
ES:DI = адрес буфера имени пула, длина этого буфера должна быть 8 байтов;
DX = индекс EMM.
На выходе: AH = байт состояния EMM.

Функция предназначена для использования в мультизадачной среде. Она позволяет нескольким процессам, работающим одновременно, использовать одни и те же именованные пулы дополнительной памяти.

11.5.12. Найти имя пула

На входе: AH = 54h;
AL = код подфункции:
0 - получить каталог пулов;
1 - найти пул по имени;
2 - определить количество открытых пулов;

	DS:SI	=	адрес буфера имени пула для подфункции 1, длина этого буфера должна быть 8 байтов;
	ES:DI	=	адрес массива 10-байтовых элементов; в первое слово элемента будет записан индекс пула, в остальные 8 - имя пула.
На выходе:	AH	=	байт состояния EMM;
	AL	=	количество элементов в массиве (подфункция 0);
	DX	=	индекс найденного пула (для подфункции 1);
	BX	=	количество открытых пулов (для подфункции 2).

Эта функция позволяет определить индекс пула по его имени или получить каталог всех именованных пулов.

11.5.13. Отобразить страницу и перейти по адресу

На входе:	AH	=	55h;
	AL	=	код подфункции: 0 - использовать массив номеров физических страниц; 1 - использовать массив сегментных адресов;
	DS:SI	=	адрес структуры MapAndJump длиной 9 байтов.
На выходе:	AH	=	байт состояния EMM.

Эта функция предназначена для перекачки страниц исполняемого кода в память и последующего выполнения этого кода.

Первые четыре байта структуры MapAndJump содержат смещение и сегментный адрес, по которым должен быть выполнен переход. Следующий байт - количество элементов в таблице отображения. Последние 4 байта содержат FAR-адрес таблицы отображения, состоящей из 4-байтовых элементов. Первое слово элемента таблицы отображения - номер логической страницы, второе - номер физической страницы.

11.5.14. Отобразить страницу и вызвать процедуру

На входе:	AH	=	56h;
	AL	=	код подфункции: 0 - использовать массив номеров физических страниц; 1 - использовать массив сегментных адресов; 2 - получить размер стека, необходимого для использования подфункций 0 и 1;
	DS:SI	=	адрес структуры MapAndCall длиной 22 байта.
На выходе:	AH	=	байт состояния EMM;
	BX	=	требуемый размер стека (заполняется при выполнении подфункции 2).

Функция работает аналогично предыдущей, но не передает управление исполняемому коду, а вызывает его как процедуру.

Первые 9 байтов структуры MapAndCall соответствуют структуре MapAndJump. Далее идет еще один байт длины таблицы отображения и 4 байта адреса другой таблицы отображения. Вторая таблица описывает отображение страниц, которое будет установлено после вызова процедуры. Последние 8 байтов структуры зарезервированы для дальнейшего использования.

11.5.15. Переслать/обменять область памяти

На входе:	AH	=	57h;
	AL	=	код подфункции: 0 - переслать область памяти; 1 - обменять область памяти;
	DS:SI	=	адрес структуры MoveInfo длиной 18 байтов.
На выходе:	AH	=	байт состояния EMM.

Функция предназначена для выполнения перемещения или обмена содержимого областей стандартной или дополнительной памяти. Возможно перекрытие исходной и результирующей облас-

тей памяти. Максимальный размер блоков, над которыми эта функция может выполнять операции - 1 мегабайт.

Структура MoveInfo содержит всю необходимую информацию о расположении блоков памяти:

Смещение	*Размер*	*Описание*
(+0)	4	Размер блока в байтах
(+4)	1	Тип исходной памяти: 0 - стандартная, 1 - EMS
(+5)	2	Индекс исходной памяти: 0 для стандартной памяти, индекс пула для EMS
(+7)	2	Смещение для исходной памяти (внутри сегмента или страницы)
(+9)	2	Адрес исходного сегмента или номер для исходной страницы
(+11)	1	Тип результирующей памяти: 0 - стандартная, 1 - EMS
(+12)	2	Индекс результирующей памяти: 0 для стандартной памяти, индекс пула для EMS
(+14)	2	Смещение для результирующей памяти (внутри сегмента или страницы)
(+16)	2	Адрес результирующего сегмента или номер для исходной страницы

11.5.16. Получить массив адресов отображения

На входе: AH = 58h;
AL = код подфункции:
0 - получить массив отображения;
1 - получить размер массива отображения;
ES:DI = адрес буфера для массива отображения.
На выходе: AH = байт состояния EMM;

CX = количество элементов в массиве отображения (для подфункции 1).

Массив отображения, получаемый при помощи этой функции, состоит из 4-байтовых элементов. Первое слово элемента содержит адрес сегмента, второе - номер физической страницы, соответствующей этому адресу.

11.6. Коды ошибок

Все функции EMM возвращают код ошибки в регистре AH:

Код	*Ошибка*
00h	Нет ошибки, нормальное завершение
80h	Внутренняя ошибка драйвера EMM
81h	Ошибка аппаратуры EMS-памяти
82h	EMM занят
83h	Неправильный индекс пула
84h	Неправильный номер запрошенной функции
85h	Больше нет доступных индексов пулов
86h	Ошибка при выполнении сохранения или восстановления контекста отображения
87h	Запрошено больше памяти, чем общее количество доступной EMS-памяти
88h	Запрошено больше страниц, чем доступно
89h	Нельзя открыть индекс пустого пула
8Ah	Пул не содержит так много страниц
8Bh	Неправильное отображение, заданы номера физических страниц, отличные от 0 - 3
8Ch	Переполнена область сохранения контекста отображения
8Dh	Многократное сохранение контекста для одного пула
8Eh	Попытка восстановления несохраненного контекста
8Fh	Неправильный номер подфункции в регистре AL
90h	Неправильный тип атрибута
91h	Не поддерживается неразрушаемая память
92h	Произошло перекрытие исходной и результирующей областей (это не ошибка, а предупреждение)
93h	Область назначения, заданная индексом, слишком мала

94h	Стандартная память перекрывается дополнительной памятью
95h	Слишком большое смещение при пересылке блока
96h	Слишком большой размер блока, больше 1 мегабайта
97h	Заданы одинаковые исходный и результирующий индексы
98h	Задан неправильный тип памяти (смещение 4)
A0h	Заданному имени не соответствует ни один пул
A1h	Заданное имя уже существует
A2h	Длина исходной области больше 1 мегабайта
A3h	Содержимое заданного блока данных неверно
A4h	Доступ к этой функции запрещен

11.7. Программа, использующая EMS

Приведенная ниже программа демонстрирует использование основных функций EMM:

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

void main(void);
void main(void) {

     unsigned frame, err;
     unsigned total, free;
     unsigned handle;
     char ver;

// Это сообщение будет переписано сначала из
// области стандартной памяти в область дополнительной,
// затем обратно в область основной.

     static char test_msg[] = "Тестовое сообщение для "
                              "записи в область EMS.";

     char buf[80];
     char _far *ptr;
     int i;

// Проверяем, установлен ли драйвер EMS.

     if(ems_init() != 0) {
          printf("\nДрайвер EMS не установлен.");
          exit(-1);
     }
```

```
// Если драйвер установлен, определяем его состояние.

     printf("\nДрайвер EMS установлен, состояние: %02.2X.",
          ems_stat());

// Выводим номер версии драйвера

     if((err = ems_ver(&ver)) != 0) {
          printf("\nОшибка: %02.2X", err);
          exit(-1);
     }
     printf("\nВерсия EMM: %02.2X", ver);

// Определяем сегмент окна доступа

     if((err = ems_fram(&frame)) != 0) {
          printf("\nОшибка: %02.2X", err);
          exit(-1);
     }

     printf("\nСегмент окна доступа: %04.4X",
          frame);

// Определяем общее количество страниц и
// количество доступных страниц.

     if((err = ems_page(&total, &free)) != 0) {
          printf("\nОшибка: %02.2X", err);
          exit(-1);
     }

     printf("\nОбщее количество страниц EMS: %d"
               "\nКоличество доступных страниц: %d",
               total, free);

// Заказываем пул в дополнительной памяти.

     if((err = ems_open(free, &handle)) != 0) {
          printf("\nОшибка: %02.2X", err);
          exit(-1);
     }

// Отображаем нулевую физическую страницу
// на нулевую логическую страницу пула.

     if((err = ems_map(0, 0, handle)) != 0) {
          printf("\nОшибка: %02.2X", err);
          exit(-1);
     }

// Конструируем указатель на физическую страницу.

     ptr = FP_MAKE(frame,0);

// Копируем текстовое сообщение в дополнительную память.
```

```
     printf("\nКопируем в EMS: %s",test_msg);

     for(i=0; test_msg[i] != 0; i++)
          ptr[i] = test_msg[i];

// Теперь копируем это сообщение обратно в стандартную память.

     for(i=0; ptr[i] != 0; i++)
          buf[i] = ptr[i];
     buf[i] = 0;

// Выводим сообщение на экран для проверки.

     printf("\nСкопировано из EMS: %s",buf);

// Закрываем пул.

     if((err = ems_clos(&handle)) != 0) {
          printf("\nОшибка: %02.2X", err);
          exit(-1);
     }

     exit(0);
}
```

Глава 12

АРИФМЕТИЧЕСКИЙ СОПРОЦЕССОР

Последнее устройство, которое мы опишем в этом томе, - арифметический сопроцессор Intel 8087/80287/80387. Он подключен непосредственно к центральному процессору и предназначен для выполнения операций над числами в формате с плавающей точкой (вещественные числа) и длинными целыми числами.

Арифметический сопроцессор значительно (в десятки раз) ускоряет вычисления, связанные с вещественными числами. Он может вычислять такие функции, как синус, косинус, тангенс, логарифмы и т. д. Разумеется, что с помощью сопроцессора можно выполнять и простейшие арифметические операции - сложения, вычитания, умножения и деления.

Основная область применения арифметического сопроцессора - научные расчеты и машинная графика. Некоторые пакеты САПР, например Autocad версии 10, отказываются работать, если в машине отсутствует сопроцессор. Более современный процессор Intel 80486 содержит встроенный арифметический сопроцессор, совместимый с 80387 (и даже немного более мощный).

Сопроцессор запускается центральным процессором. После запуска он выполняет все вычисления самостоятельно и параллельно с работой центрального процессора. Если центральный процессор выдает очередную команду сопроцессору в момент времени, когда тот еще не закончил выполнение предыдущей команды, центральный процессор переводится в состояние ожидания. Если же сопроцессор ничем не занят, центральный процессор, выдав команду сопроцессору, продолжает свою работу, не дожидаясь завершения вычисления. Впрочем, есть специальные средства синхронизации (команда FWAIT).

Как программировать сопроцессор?

Команды, предназначенные для выполнения сопроцессором, записываются в программе как обычные машинные команды центрального процессора. Но все эти команды начинаются с байта, соответствующего команде центрального процессора ESC.

Встретив такую команду, процессор передает ее на выполнение сопроцессору, а сам продолжает выполнение программы со следующей команды.

Ассемблерные мнемоники всех команд сопроцессора начинаются с буквы F, например: FADD, FDIV, FSUB и т. д. Команды сопроцессора могут адресоваться к операндам, аналогично обычным командам центрального процессора. Операндами могут быть либо данные, расположенные в основной памяти компьютера, либо внутренние регистры сопроцессора.

Возможны все виды адресации данных, используемые центральным процессором.

Прежде чем начать обсуждение команд, выполняемых сопроцессором, приведем форматы используемых данных. Как мы уже говорили, сопроцессор может работать либо с данными в формате с плавающей точкой, либо с целыми числами. В следующем разделе мы рассмотрим используемые форматы чисел с плавающей точкой или форматы вещественных чисел.

12.1. Вещественные числа

Прежде чем говорить о форматах вещественных числе, используемых сопроцессором, вспомним о числах с плавающей точкой, встречающихся в научных расчетах. В общем виде эти числа можно записать следующим образом:

(знак)(мантисса)*10(знак)(порядок)

Например: -1.35*105

Здесь знак - это минус, мантисса - 1.35, порядок - 5. Порядок тоже может иметь знак. В этом представлении чисел для Вас вряд ли есть что-либо новое. Вспомним также такое понятие, как нормализованное представление чисел:

> *если целая часть мантиссы числа состоит из одной, не равной нулю, цифры, то число с плавающей точкой называется нормализованным.*

Преимущество использования нормализованных чисел в том, что для фиксированной разрядной сетки числа (для фиксированного количества цифр в числе) такие числа имеют наибольшую точность. Кроме того, нормализованное представление исключает

неоднозначность - каждое число с плавающей точкой может быть представлено различными (ненормализованными) способами:

```
123.5678*105 = 12.35678*106 = 1.235678*107 = 0.1235678*108
```

Для тех, кто программировал на языках высокого уровня, знакомо следующее представление чисел с плавающей точкой:

```
(знак)(мантисса)E(знак)(порядок)
```

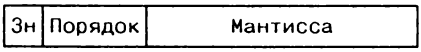
Например, -5.35E-2 означает число -5.35*10-2. Такое представление называется научной нотацией.

Сопроцессор 8087/80287/80387 может работать с вещественными числами в трех форматах:

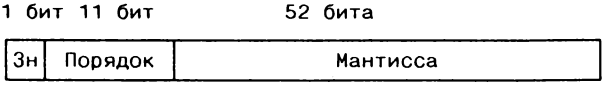
- одинарной точности;
- двойной точности;
- расширенной точности.

Эти числа занимают в памяти соответстсвенно 4, 8 и 10 байтов:

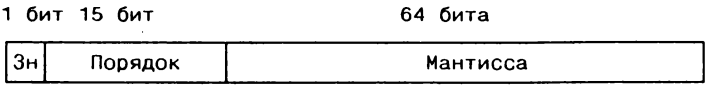
Одинарная точность



Двойная точность



Расширенная точность



В любом представлении старший бит "Зн" определяет знак вещественного числа:

0 - положительное число;
1 - отрицательное число.

Все равные по абсолютному значению положительные и отрицательные числа отличаются только этим битом. В остальном числа с разным знаком полностью симметричны. Для представления

отрицательных чисел здесь не используется дополнительный код, как это сделано в центральном процессоре.

Арифметический сопроцессор работает с нормализованными числами, поэтому поле мантиссы содержит мантиссу нормализованного числа.

Здесь используется двоичное представление чисел. Сформулируем определение нормализованного числа для этого случая:

> *если целая часть мантиссы числа в двоичном представлении равна 1, то число с плавающей точкой называется нормализованным.*

Так как для нормализованного двоичного числа целая часть всегда равна единице, то эту единицу можно не хранить. Именно так и поступили разработчики арифметического сопроцессора - в форматах одинарной и двойной точности целая часть мантиссы не хранится. Таким образом экономится один бит памяти.

Для наглядности представим мантиссу числа в такой форме:

```
n.nnnnnnnnnn...n
```

Здесь символом 'n' обозначается либо 0, либо 1. Нормализованные числа в самой левой позиции содержат 1, поэтому их можно изобразить еще и в таком виде:

```
1.nnnnnnnnnn...n
```

Представление с расширенной точностью используется сопроцессором для выполнения всех операций. И даже более - все операции с числами сопроцессор выполняет над числами только в формате с расширенной точностью. В этом формате хранится и "лишний" бит целой части нормализованного числа.

Основная причина использования для вычислений расширенной точности - предохранение программы от возможной потери точности вычислений, связанной с большими различиями в порядках чисел, участвующих в арифметических операциях.

Поле порядка - это степень числа 2, на которую умножается мантисса, плюс смещение, равное 127 для одинарной точности, 1023 - для двойной и 16383 - для расширенной точности.

Для того чтобы определить абсолютное значение числа с плавающей точкой, можно воспользоваться следующими формулами:

- одинарная точность: 1.(цифры мантиссы)*2(P-127)
- двойная точность: 1.(цифры мантиссы)*2(P-1023)
- расширенная точность: 1.(цифры мантиссы)*2(P-16383)

Знак числа, как уже говорилось, определяется старшим битом.

Приведем конкретный пример. Пусть мы имеем число с одинарной точностью, которое в двоичном виде выглядит так:

```
1 01111110 11000000000000000000000
```

Для этого числа знаковый бит равен 1 (отрицательное число), порядок - 126, мантисса - 11 (в двоичной системе счисления).

Значение этого числа равно:

```
-1.11 * 2(126-127) = -1.75 * 2-1 = -0,875
```

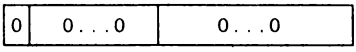
Рассмотрим особые случаи представления вещественных чисел.

- Нуль - это число, у которого порядок и мантисса равны нулю. Нуль может иметь положительный или отрицательный знаки, которые игнорируются в операциях сравнения. Таким образом, имеется два нуля - положительный и отрицательный.
- Наименьшее положительное число - это число, которое имеет нулевой знаковый бит, значение порядка, равное единице, и значение мантиссы, равное нулю. В зависимости от представления наименьшее положительное число имеет следующие значения: 1,17*10-38 (одинарная точность), 2.23*10-308 (двойная точность), 3.37*10-4932 (расширенная точность).
- Наибольшее отрицательное число - это число, полностью совпадающее с наименьшим положительным числом, но имеющее бит знака, установленный в 1.
- Наибольшее положительное число - это число, которое имеет нулевой знаковый бит, поле порядка, в котором все биты, кроме самого младшего, равны 1, и содержит единицы во всех разрядах мантиссы. В зависимости от представления наибольшее положительное число имеет следующие значения: 3.37*1038 (одинарная точность), 1.67*10308 (двойная точность), 1.2*104932 (расширенная точность).

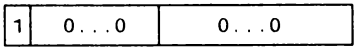
- Наименьшее отрицательное число - это число, полностью совпадающее с наибольшим положительным числом, но имеющее бит знака, установленный в 1.
- Положительная и отрицательная бесконечность - это число, которое содержит все единицы в поле порядка и все нули в поле мантиссы. В зависимости от состояния знакового бита может быть положительная и отрицательная бесконечность. Бесконечность может получиться, например, как результат деления конечного числа на нуль.
- Нечисло - содержит все единицы в поле порядка и любое значение в поле мантиссы. Нечисло может возникнуть в результате выполнения неправильной операции при замаскированных особых случаях (ошибкам при работе с сопроцессором посвящен отдельный раздел этой главы).
- Неопределенность - содержит в поле порядка все единицы, а в поле мантиссы - число 1000..0 (для одинарной и двойной точности) или 11000..0 (для расширенной точности, т. к. в этом формате хранится старший бит мантиссы).

Для большей наглядности сведем все возможные представления вещественных чисел в таблицу:

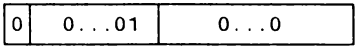
Положительный нуль



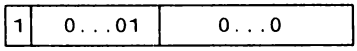
Отрицательный нуль



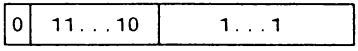
Наименьшее положительное число



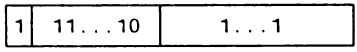
Наибольшее отрицательное число



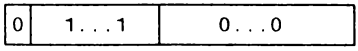
Наибольшее положительное число



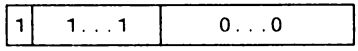
Наименьшее отрицательное число



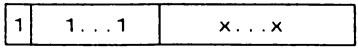
Положительная бесконечность



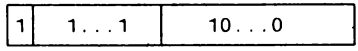
Отрицательная бесконечность



Нечисло



Неопределенность



12.2. Целые числа

Арифметический сопроцессор наряду с вещественными числами способен обрабатывать целые числа. Он имеет команды, выполняющие преобразования целых чисел в вещественные и обратно.

Возможно четыре формата целых чисел:

- целое число;
- короткое целое число;
- длинное целое число;
- упакованное десятичное число.

Целое число занимает два байта. Его формат полностью соответствует используемому центральным процессором. Для представления отрицательных чисел используется дополнительный код. Короткое целое и длинное целое числа имеют аналогичные форматы, но занимают, соотвесттсвенно 4 и 8 байтов.

Упакованное десятичное число занимает 10 байтов. Это число содержит 18 десятичных цифр, расположенных по две в каждом байте. Знак упакованного десятичного числа находится в старшем бите самого левого байта. Остальные биты старшего байта должны быть равны нулю.

Существуют команды сопроцессора, которые преобразуют числа в формат упакованных десятичных чисел из внутреннего представления в расширенном вещественном формате. Если программа делает попытку преобразования в упакованный формат денормализованных чисел, нечисел, бесконечности и т. п., в результате получается неопределенность. Неопределенность в упакованном формате представляет из себя число, в котором два старших байта содержат единицы во всех разрядах. Содержимое остальных восьми байтов произвольно. При попытке использовать такое упакованное число в операциях фиксируется ошибка.

Мы подробно рассмотрели формат представления вещественных чисел и отметили, что в этом формате для представления отрицательных чисел используется специальный знаковый бит. Для целых чисел используется дополнительный код.

В дополнительном коде положительные числа содержат нуль в самом старшем бите числа:

```
0XXX XXXX XXXX XXXX
```

Для получения отрицательного числа в дополнительном коде из положительного надо проинвертировать каждый бит числа и затем прибавить к числу единицу. Например, число +5 в дополнительном коде выглядит следующим образом:

```
0000 0000 0000 0101 = +5
```

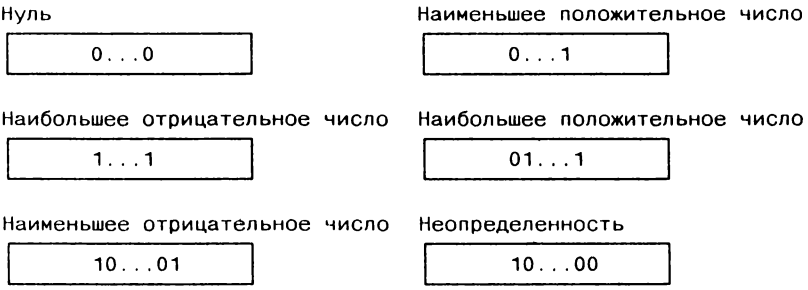
Для получения числа -5 вначале проинвертируем значение каждого бита:

```
1111 1111 1111 1010
```

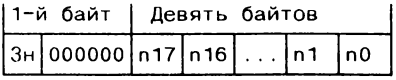
Теперь прибавим к полученному числу +1:

```
1111 1111 1111 1011 = -5
```

Приведем возможные варианты представления целых чисел:



Упакованное десятичное число имеет следующий вид:



На этом рисунке n0...n17 означают разряды десятичного числа. Они могут изменяться в пределах от 0000 до 1001, т. е. от 0 до 9 в десятичной системе счисления.

Теперь, после того как мы рассмотрели форматы данных, с которыми может работать арифметический сопроцессор, можно перейти к изучению внутренних регистров сопроцессора.

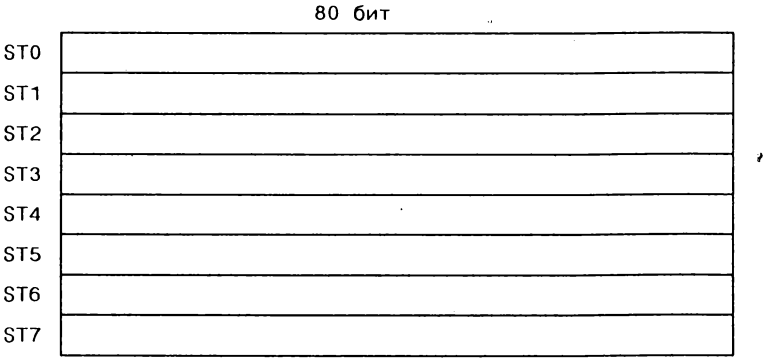
12.3. Регистры сопроцессора

Арифметический сопроцессор содержит восемь численных 80-битовых регистров, предназначенных для хранения промежуточных результатов вычислений, регистра управления, регистра со-

стояния, регистра тегов, регистра указателя команды и регистра указателя операнда.

12.3.1. Численные регистры

Мы будем обозначать численные регистры как ST0 - ST7. Они приведены на следующем рисунке:



Численные регистры используются как стек. Регистр состояния в поле ST содержит номер численного регистра, являющегося вершиной стека. При выполнении команд в качестве операнда могут выступать численные регистры. В этом случае номер указанного в команде регистра прибавляется к содержимому поля ST регистра состояния и таким образом определяется используемый регистр. Большинство команд после выполнения увеличивают поле ST регистра состояния, как бы записывая результаты своей работы в стек численных регистров. Вы можете использовать регистры как массив, но в этом случае необходимо заботится о постоянстве поля ST регистра состояния, т. к. в противном случае номера численных регистров будут изменяться.

12.3.2. Регистр тегов

Этот регистр разделен на восемь двухбитовых полей, которые мы обозначим как TAG0...TAG7. Каждое поле относится к своему численному регистру:

TAG0	TAG1	TAG2	TAG3	TAG4	TAG5	TAG6	TAG7

Поля регистра тегов классифицируют содержимое "своего" численного регистра:

00 регистр содержит действительное ненулевое число;
01 в регистре находится нуль;
10 регистр содержит недействительное число (нечисло, бесконечность, неопределенность);
11 пустой неинициализированный регистр.

Например, если все регистры сопроцессора были пустые, а затем в стек численных регистров было занесено одно действительное ненулевое значение, содержимое регистра тегов будет 3FFFh.

12.3.3. Регистр управления

Регистр управления для сопроцессора 8087 показан на следующем рисунке:

15-13	12	11-10	9-8	7	6	5	4	3	2	1	0
XXXXXXXX	IC	RC	PC	IEM	XXX	PM	UM	OM	ZM	DM	IM

Регистр управления сопроцессоров 80287/80387 и сопроцессора, входящего в состав процессора 80486, имеет аналогичный формат, за исключением того, что бит 7 в нем не используется:

15-13	12	11-10	9-8	7-6	5	4	3	2	1	0
XXXXXXXX	IC	RC	PC	XXXXXXXX	PM	UM	OM	ZM	DM	IM

Биты 0...5 - маски особых случаев. Особые случаи иногда возникают при выполнении команд сопроцессора, например при делении на нуль, переполнении и т. д. Если все биты масок особых случаев равны нулю, особый случай вызывает прерывание центрального процессора INT 10h (обратите внимание, что это прерывание используется BIOS для работы с дисплейным адаптером). Если же особые случаи замаскированы установкой соответствующих битов в единичное состояние, прерывание не вырабатывается, а в качестве результата возвращается особое значение - бесконечность, нечисло и т. д.

Приведем таблицу масок особых случаев:

IM	маска недействительной операции;
DM	маска денормализованного результата;
ZM	маска деления на нуль;
OM	маска переполнения;
UM	маска антипереполнения;
PM	маска особого случая при неточном результате;
IEM	маскирование одновременно всех особых случаев вне зависимости от установки битов 0...5 регистра управления, этот бит действителен только для сопроцессора 8087.

Подробнее особые случаи и условия их возникновения будут описаны позже, когда мы займемся ошибками при выполнении команд в сопроцессоре.

Поле PC управляет точностью вычислений в сопроцессоре:

00	использование расширенной точности, этот режим устанавливается при инициализации сопроцессора;
10	округление результата до двойной точности;
00	округление результата до одинарной точности.

Искусственное ухудшение точности вычислений не приводит к ускорению работы программы. Режимы с пониженной точностью предназначены для эмуляции процессоров, использующих двойную и одинарную точность, соответственно.

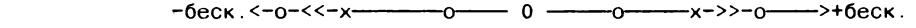
Двухбитовое поле RC задает режим округления при выполнении операций с вещественными числами:

00	округление к ближайшему числу, этот режим устанавливается при инициализации сопроцессора;
01	округление в направлении к отрицательной бесконечности;
10	округление в направлении к положительной бесконечности;
11	округление в направлении к нулю.

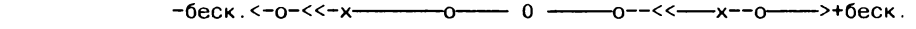
На следующих рисунках демонстрируются перечисленные выше режимы округления. Символами 'o' обозначены точные значе-

ния вещественных чисел, символами 'x' - приближенные значения. Стрелки '<<' и '>>' указывают направление округления. В центре линии расположен нуль числовой оси, на ее левом и правом конце - отрицательная и положительная бесконечности.

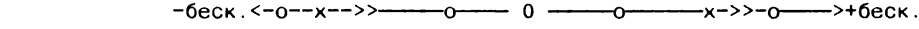
Округление в направлении к ближайшему числу.



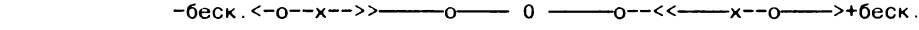
Округление в направлении к отрицательной бесконечности.



Округление в направлении к положительной бесконечности.



Округление в направлении к нулю.



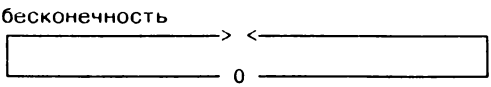
Для наибольшего уменьшения ошибок вычислений наиболее целесообразно использовать режим округления в направлении к ближайшему числу. Режим округления в направлении к нулю используется при моделировании целочисленной арифметики.

Остальные два режима округления используют в интервальной арифметике. Для получения наиболее точного результата каждая команда (операция) выполняется два раза - первый раз с округлением в направлении к отрицательной бесконечности, второй раз - в направлении к положительной бесконечности. Точный результат лежит между полученными значениями. Заметьте, что здесь речь идет только об отдельных операциях, но не о том, чтобы выполнить всю программу вычислений вначале с одним режимом округления, а затем с другим.

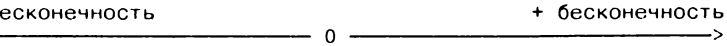
Поле IC регистра управления предназначено для управления бесконечностью:

0 - проективный режим;
1 - аффинный режим.

В проективном режиме существует только одна бесконечность, она не имеет знака:



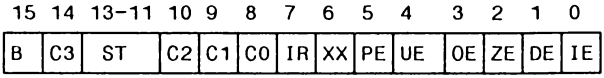
В афинном режиме имеется две бесконечности - положительная и отрицательная:



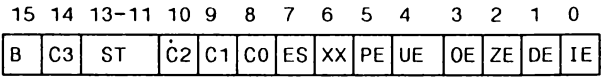
Афинный режим допускает выполнение многих операций с бесконечностями - сложение, умножение и т. д.

12.3.4. Регистр состояния

Поля регистра состояния сопроцессора 8087 показаны на следующем рисунке:



Регистр состояния сопроцессоров 80287/80387 и сопроцессора, входящего в состав процессора 80486, имеет другой формат:



В обоих форматах биты 0...5 - флажки особых случаев, устанавливаемые всегда при возникновении особых случаев, даже замаскированных установкой в единицу соответствующих битов регистра управления. Приведем таблицу флажков особых случаев:

IE недействительная операция;
DE денормализованный результат;
ZE деление на нуль;
OE переполнение;
UE антипереполнение;
PE неточный результат.

Для того чтобы сбросить установившийся флажок, программа должна явным образом установить его в нуль, выполнив команду записи в регистр состояния.

Назначение бита 7 регистра состояния различно для сопроцессора 8087 и сопроцессоров 80287/80387. Для сопроцессора 8087 этот бит обозначается IR и содержит флаг запроса прерывания

при возникновении незамаскированного особого случая. В этом случае флаг устанавливается в 1.

Сопроцессоры 80287/80387 используют бит 7 в качестве флага суммарной ошибки, который устанавливается в 1 при возникновении незамаскированного особого случая.

Биты C0, C1, C2, C3 - это коды условий. Они определяются по результату выполнения команд сравнения и команды нахождения остатка. Мы расскажем о них при описании соответствующих команд сопроцессора.

Поле ST занимает три бита 11...13 и содержит номер регистра, являющегося вершиной стека численных регистров.

Бит B - бит занятости. Он устанавливается в 1, когда процессор выполняет команду или происходит прерывание от сопроцессора. Если сопроцессор свободен, бит занятости установлен в 0.

12.3.5. Регистры указателя команды и указателя операнда

Регистры указателя команды и указателя операнда предназначены для обработки особых случаев, возникающих при выполнении команд в сопроцессоре.

В сопроцессоре 8087 указатель команды содержит 20-разрядный адрес команды, вызвавшей особый случай и код выполняемой в этот момент операции. Адрес команды здесь указывается без учета предшествующих команде префиксов:

Адрес команды (0...15)		
Адрес команды (16...19)	X	Код операции (0...10)

Сопроцессоры 80287/80387 в реальном режиме имеют такой же формат регистра указателя команд, но этот указатель показывает на первый префикс команды, вызвавшей особый случай.

Защищенный режим работы центрального процессора и сопроцессора выходит за рамки данной книги, однако для полноты изложения приведем формат указателей и для этого режима. В защищенном режиме адрес состоит из селектора (в какой-то степени соответствует сегментной компоненте адреса реального режима) и смещения. Формат указателя команды для защищенного режима представлен на следующем рисунке:

Смещение команды
Селектор команды

Код операции здесь отсутствует, но его легко получить, пользуясь адресом команды.

Если при возникновении особого случая использовался операнд, находящийся в оперативной памяти, его адрес записывается в регистр указателя операнда. Приведем форматы этого регистра для реального и защищенного режимов работы.

Формат указателя операнда для реального режима:

Адрес операнда (0...15)	
Адрес операнда (16...19)	XXXXXXXXXXXXXXXXX

Формат указателя операнда для защищенного режима:

Смещение операнда
Селектор операнда

12.4. Система команд сопроцессора

Возможны три формата команд сопроцессора, аналогичные форматам команд центральных процессоров 8086/80286/80386. Это команды с обращением к оперативной памяти, команды с обращением к одному из численных регистров и команды без операндов, заданных явным образом.

Команды с обращением к памяти могут занимать от двух до четырех байтов, в зависимости от способа адресации операнда, находящегося в памяти:

1 байт		1 байт			1 байт	1 байт
11011	КОП1	MOD	КОП2	R/M	Смещение1	Смещение2

Первые пять битов соответствуют команде центрального процессора ESC. Поля КОП1 и КОП2 определяют выполняемую команду, т. е. содержат код операции. Поля MOD и R/M вместе с

полями "Смещение1" и "Смещение2" задают адрес операнда в памяти аналогично тому, как это происходит в процессорах 8086/80286/80386. Однако есть и отличия, связанные с возможностью адресации численных регистров сопроцессора.

Приведем таблицу, показывающую зависимость способа адресации от содержимого полей MOD и R/M:

Поле R/M	Поле MOD			
	00	01	10	11
000	(bx)+(si)	(bx)+(si)+disp8	(bx)+(si)+disp16	ST0
001	(bx)+(di)	(bx)+(di)+disp8	(bx)+(di)+disp16	ST1
010	(bp)+(si)	(bp)+(si)+disp8	(bp)+(si)+disp16	ST2
011	(bp)+(di)	(bp)+(di)+disp8	(bp)+(di)+disp16	ST3
100	(si)	(si)+disp8	(si)+disp16	ST4
101	(di)	(di)+disp8	(di)+disp16	ST5
110	disp16	(bp)+disp8	(bp)+disp16	ST6
111	(bx)	(bx)+disp8	(bx)+disp16	ST7

Если в таблице указаны значения смещения disp8 или disp16, это означает, что в команде присутствуют соответственно один или два байта смещения. Если поле MOD содержит значение 11, возможна адресация численных регистров ST0...ST1. При этом команда не содержит байтов смещения.

Формат команды с обращением к численному регистру приведен на следующем рисунке:

1 байт		1 байт		
11011	КОП1	11	КОП2	STi

Видно, что это есть частный случай предыдущей команды, в которой поле MOD содержит значение 11 и отсутствуют байты смещения.

Самый простой формат имеют команды без явного обращения к операндам:

1 байт		1 байт	
11011	КОП1	11	КОП2

Разумеется, если Вы пишете программу для сопроцессора на языке ассемблера, Вы можете использовать мнемоническое обо-

значение команд. Все мнемоники команд сопроцесора начинаются с буквы F, поэтому их легко отличить от команд процессоров 8086/80286/80386/80486.

Все команды сопроцессора можно разделить на несколько групп:

- команды пересылки данных;
- арифметические команды;
- команды сравнений чисел;
- трансцендентные команды;
- управляющие команды.

Команды пересылки данных предназначены для загрузки чисел из оперативной памяти в численные регистры, записи данных из численных регистров в оперативную память, копирования данных из одного численного регистра в другой.

Арифметические команды выполняют такие операции, как сложение, вычитание, умножение, деление, извлечение квадратного корня, нахождение частичного остатка, округление и т. п.

Команды сравнения сравнивают вещественные и целые числа, выполняют анализ чисел.

Трансцендентные команды предназначены для вычисления различных тригонометрических, логарифмических, показательных и гиперболических функций - sin(), cos(), tg() и т. п.

Последняя группа команд - управляющие команды - обеспечивают установку режима работы арифметического сопроцессора, его сброс и инициализацию, перевод сопроцессора в защищенный режим работы и т. д.

Следующие разделы будут посвящены детальному описанию различных групп команд сопроцессора.

12.4.1. Команды пересылки данных

Запись в стек

FLD ST(0) <- память, вещественный формат
FILD ST(0) <- память, целый формат
FBLD ST(0) <- память, десятичный формат

Команды FLD, FILD, FBLD загружают в вершину стека соответственно вещественное, целое и десятичное числа.

При выполнении этих команд операнд считывается из оперативной памяти, преобразуется в формат с расширенной точностью. Затем поле ST регистра состояния уменьшается на единицу и выполняется запись операнда в численный регистр, определяемый новым значением поля ST, т. е. операнд записывается в стек численных регистров, а указатель стека - поле ST - уменьшается на единицу. По своему действию эти команды напоминают команду PUSH центрального процессора.

Непосредственно перед загрузкой численного регистра проверяется содержимое поля TAG0. Если это содержимое не равно 11 (пустой регистр), в регистре состояния устанавливается флаг IE (недействительная операция) и вырабатывается прерывание (если в регистре управления не установлена маска IM - маска недействительной операции).

Извлечение из стека

FSTP память -> ST(0), вещественный формат
FISTP память -> ST(0), целый формат
FBSTPпамять -> ST(0), десятичный формат

Команды извлечения чисел из стека выполняют действие, обратное только что описанному. Содержимое численного регистра, номер которого определяется полем ST регистра состояния, преобразуется в необходимый формат и записывается в ячейки оперативной памяти, заданные операндом команды.

После записи содержимое поля ST увеличивается на единицу. Эти действия аналогичны выполняемым командой POP центрального процессора.

В зависимости от команды (FSTP, FISTP или FBSTP) производится преобразование формата (соответственно из расширенного в вещественный, целый или десятичный). В процессе преобразования для команд FSTP и FISTP выполняется округление в соответствии с содержимым поля RC регистра управления. Для команды FBSTP округление всегда выполняется следующим образом: прибавляется число 0,5, затем дробная часть результата отбрасывается.

Копирование данных

FST память -> ST(0), вещественный формат
FIST память -> ST(0), целый формат
FBST память -> ST(0), десятичный формат
(только 80387, 80486)

Эти команды пересылают данные из верхушки стека в область памяти, указанную операндом команды. При этом содержимое указателя стека (поля ST) не изменяется.

Команда FST в качестве операнда может использовать ссылку на численный регистр ST(i), поэтому Вы можете использовать эту команду для копирования верхушки стека в любой другой численный регистр.

При записи данных в оперативную память выполняется преобразование формата (в вещественный для FST, в целый для FIST и в десятичный для FBST).

Для сопроцессора 80286 вместо отсутствующей команды FBST можно выполнить следующие две команды, которые приведут к такому же результату:

```
FLD     ST(0)
FBSTP     dec_number
```

Обмен

FXCH ST(i) -> ST(0), ST(0) -> ST(i)

Команда выполняет обмен содержимым верхушки стека ST(0) и численного регистра, указанного в качестве операнда команды.

Загрузка констант

FLDZ	0 -> ST(0) - загрузить нуль
FLD1	1 -> ST(0) - загрузить единицу
FLDPI	"Пи" -> ST(0) - загрузить число "пи".
FLDLG2	log102 -> ST(0) - загрузить log102
FLDLN2	loge2 -> ST(0) - загрузить loge2
FLDL2T	loge10 -> ST(0) - загрузить loge10
FLDL2E	log2e -> ST(0) - загрузить log2e

Гораздо быстрее загружать константы с помощью специальных команд, чем использовать команды загрузки данных из оперативной памяти.

12.4.2. Арифметические команды

Сопроцессор использует шесть основных типов арифметических команд:

Fxxx	Первый операнд берется из верхушки стека (источник), второй - следующий элемент стека. Результат выполнения команды записывается в стек.
Fxxx память	Источник берется из памяти, приемником является верхушка стека ST(0). Указатель стека ST не изменяется, команда действительна лишь для операндов с одинарной и двойной точностью.
FIxxx память	Аналогично предыдущему типу команды, но операндами могут быть 16- или 32-битовые целые числа.
Fxxx ST, ST(i)	Для этого типа регистр ST(i) является источником, а ST(0) - верхушка стека - приемником. Указатель стека не изменяется.
Fxxx ST(i), ST	Для этого типа регитр ST(0) является источником, а ST(i) - приемником. Указатель стека не изменяется.
FxxxP ST(i), ST	Регистр ST(i) - приемник, регистр ST(0) - источник. После выполнения команды источник ST(0) извлекается из стека.

Строка "xxx" может принимать следующие значения:

ADD	Сложение
SUB	Вычитание
SUBR	Обратное вычитание, уменьшаемое и вычитаемое меняются местами
MUL	Умножение
DIV	Деление
DIVR	Обратное деление, делимое и делитель меняются местами

Кроме основных арифметических команд имеются дополнительные арифметические команды:

FSQRT	Извлечение квадратного корня
FSCALE	Масштабирование на степень числа 2
FPREM	Вычисление частичного остатка

FRNDINT	Округление до целого
FXTRACT	Выделение порядка числа и мантиссы
FABS	Вычисление абсолютной величины числа
FCHS	Изменение знака числа

По команде FSQRT вычисленное значение квадратного корня записывается в верхушку стека ST(0).

Команда FSCALE изменяет порядок числа, находящегося в ST(0). По этой команде значение порядка числа ST(0) складывается с масштабным коэффициентом, который должен быть предварительно записан в ST(1). Действие этой команды можно представить следующей формулой:

```
ST(0) = ST(0) * 2n, где -215 <= n <= +215
```

В этой формуле n - это ST(1).

Команда FPREM вычисляет остаток от деления делимого ST(0) на делитель ST(1). Знак результата равен знаку ST(0), а сам результат получается в вершине стека ST(0).

Действие команды заключается в сдвигах и вычитаниях, аналогично "ручному" делению "в столбик". После выполнения команды флаг C2 регистра состояния может принимать следующие значения:

0 - остаток от деления, полученный в ST(0), меньше делителя ST(1), команда завершилась полностью;

1 - ST(0) содержит частичный остаток, программа должна еще раз выполнить команду для получения точного значения остатка.

Команда RNDINT округляет ST(0) в соответствии с содержимым поля RC управляющего регистра.

Команда FABS вычисляет абсолютное значение ST(0). Аналогично команда FCHS изменяет знак ST(0) на противоположный.

12.4.3. Команды сравнений чисел

В процессорах 8086/80286/80386 команды условных переходов выполняются в соответствии с установкой отдельных битов регистра флагов процессора. В арифметическом сопроцессоре существуют специальные команды сравнений, по результатам выполне-

ния которых устанавливаются биты кодов условий в регистре состояния:

FCOM	Сравнение
FICOM	Целочисленное сравнение
FCOMP	Сравнение и извлечение из стека
FICOMP	Целочисленное сравнение и извлечение из стека
FCOMPP	Сравнение и двойное извлечение из стека
FTST	Сравнение операнда с нулем
FXAM	Анализ операнда

Команда FCOM вычитает сожержимое операнда, размещенного в оперативной памяти, из верхушки стека ST(0). Результат вычитания никуда не записывается, и указатель верхушки стека ST не изменяется.

Обозначим операнд команды сравнения как "x". В следующей таблице приведем значения битов кодов условия после выполнения команды "FCOM x":

C3	*C0*	*Условие*
0	0	ST(0) > x
0	1	ST(0) < x
1	0	ST(0) = x
1	1	ST(0) и x не сравнимы

Последняя комбинация возникает при попытке сравнения нечисел, неопределенностей или бесконечностей, а также в некоторых других случаях.

Команда FICOM работает с 16- или 32-битовыми числами, в остальном она аналогична команде FCOM.

Команды FCOMP и FICOMP аналогичны соответственно командам FCOM и FICOM, за исключением того, что после выполнения операнд извлекается из стека.

Команда FCOMPP выполняет те же действия, что и FCOM, но она после выполнения извлекает из стека оба операнда, участвовавших в сравнении.

Для сравнения операнда с нулем предназначена команда FTST. После ее выполнения коды условий устанавливаются в соответствии со следующей таблицей:

C3	*C0*	*Условие*
0	0	ST(0) > 0
0	1	ST(0) < 0
1	0	ST(0) = 0
1	1	ST(0) и 0 не сравнимы

Команда FXAM анализирует содержимое ST(0). После ее выполнения устанавливаются коды условий, по которым можно судить о знаке числа, о его конечности или бесконечности, нормализованности и т. д.

Бит C1 содержит знак анализируемого числа:

C1	*Знак числа*
0	положительное число;
1	отрицательное число.

С помощью бита C0 можно определить, является число конечным или бесконечным:

C0	*Конечность/бесконечность числа*
0	конечное число;
1	бесконечное число.

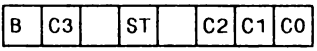
Для конечных чисел дальнейшая классификация может проводиться по содержимому кодов условий C2 и C3:

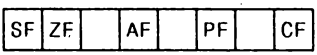
C3	*C2*	*Описание числа*
0	0	Ненормализованное число
0	1	Нормализованное число
1	0	Нулевое число
1	1	Число денормализовано

Аналогично для бесконечных чисел коды условий C2 и C3 имеют следующее значение:

C3	*C2*	*Описание числа*
0	0	Нечисло
0	1	Бесконечное число
1	0	Пустое число
1	1	-"-

С помощью команды FSTSW AX программа может переписать содержимое регистра состояния сопроцессора в регистр AX центрального процессора. Далее содержимое регистра AH можно переписать в регистр флагов центрального процессора при помощи команды SAHF. Биты кодов условий сопроцессора отображаются на регистр флагов центрального процессора так, что для анализа кодов условий можно использовать команды условных переходов:





Например, в следующем фрагменте программы выполняется переход к метке error, если операнды несравнимы:

```
fcom
fstsw     ax
sahf
je          error
```

12.4.4. Трансцендентные команды

Трансцендентные команды предназначены для вычисления таких функций, как тригонометрические (sin, cos, tg,...), обратные тригонометрические (arcsin, arccos,...), показательные (xy, 2x, 10x, ex), гиперболические (sh, ch, th,...), обратные гиперболические (arsh, arch, arcth,...). Ниже приведены все трансцендентные команды сопроцессора:

FPTAN	Вычисление частичного тангенса
FPATAN	Вычисление частичного арктангенса
FYL2X	Вычисление y*log2(x)
FYL2XP1	Вычисление y*log2(x+1)
F2XM1	Вычисление 2x-1
FCOS	Вычисление cos(x) (только 80387/80486)
FSIN	Вычисление sin(x) (только 80387/80486)
FSINCOS	Вычисление sin(x) и cos(x) одновременно (только 80387/80486)

Команда FPTAN вычисляет частичный тангенс ST(0), размещая в стеке такие два числа x и y, что y/x = tg(ST(0)). После

выполнения команды число y располагается в ST(0), а число x включается в стек сверху (т. е. записывается в ST(1)). Аргумент команды FPTAN должен находиться в пределах:

```
0 <= ST(0) <= pi/4
```

По полученному значению частичного тангенса можно вычислить другие тригонометрические функции, используя формулы:

```
sin(z) = 2*(y/x) / (1 + (y/x)2)
cos(z) = (1 - (y/x)2) / (1 + (y/x)2)
tg(z/2) = y/x;
ctg(z/2) = x/y;
cosec(z) =  (1 + (y/x)2) / 2*(y/x)
sec(z) = (1 + (y/x)2) / (1 - (y/x)2)
```

В этих формулах z - значение, находившееся в ST(0) до выполнения команды FPTAN, x и y - значения в регистрах ST(0) и ST(1) соответственно.

Команда FPATAN вычисляет частичный арктангенс

```
z = arctg(ST(0)/ST(1)) = arctg(x/y).
```

Перед выполнением команды числа x и y располагаются в ST(0) и ST(1), сответственно. Аргументы команды FPATAN должен находится в пределах:

```
0 < y < x
```

Результат записывается в ST(0).

Команда FYL2X вычисляет выражение y*log2(x), операнды x и y размещаются соответственно в ST(0) и ST(1). Операнды извлекаются из стека, а результат записывается в стек, параметр x должен быть положительным числом. Пользуясь результатом выполнения этой команды, можно вычислить следующим образом логарифмические функции:

```
log2(x) = FYL2(x)
loge(x) = loge(2) * log2(x) = FYL2X(loge(2), x) =
          = FYL2X(FLDLN2, x)
log2(x) = log10(2) * log2(x) = FYL2X (log10(2), x) =
          = FYL2X(FLDLG2, x)
```

Функция FYL2XP1 вычисляет выражение y*log2(x+1), где x соответствует ST(0), а y - ST(1). Результат записывается в ST(0), оба операнда выталкиваются из стека и теряются.

На операнд x накладывается ограничение:

```
0, < x < 1 - 1/sqrt(2)
```

Команда F2XM1 вычисляет выражение 2x-1, где x - ST(0). Результат записывается в ST(0), параметр x ограничен:

```
0 <= x <= 0,5
```

Команда FCOS вычисляет cos(x) (только для 80387/80486). Параметр x должен находиться в ST(0), туда же записывается результат выполнения команды.

Команда FSIN аналогична команде FCOS, но вычисляет значение косинуса ST(0).

Команда FSINCOS вычисляет одновременно значения синуса и косинуса параметра ST(0). Значение синуса записывается в ST(1), косинуса - в ST(0).

На этом мы закончим описание трансцендентных команд сопроцессора и перейдем к управляющим командам.

12.4.5. Управляющие команды

Управляющие команды предназначены для работы с нечисловыми регистрами сопроцессора. Некоторые команды имеют альтернативные варианты. Мнемоники этих команд могут начинаться с FN или с F. Первый вариант соответстсвует командам "без ожидания"; для них процессор не проверяет, занят ли сопроцессор выполнением команды, т. е. бит занятости B не проверяется. Численные особые случаи также игнорируются. Команды "с ожиданием" действуют так же, как и обычные команды сопроцессора.

Приведем таблицу управляющих команд сопроцессора:

FNSTCW (FSTCW)	Записать управляющее слово
FLDCW	Загрузить управляющее слово
FNSTSW (FSTSW)	Записать слово состояния
FNSTSW AX (FSTSW AX)	Записать слово состояния в AX, не поддерживается сопроцессором 8087
FNCLEX (FCLEX)	Сбросить особые случаи
FNINIT (FINIT)	Инициализировать сопроцессор
FNSTENV (FSTENV)	Записать среду
FLDENV	Загрузить среду
FNSAVE (FSAVE)	Записать полное состояние
FRSTOR	Восстановить полное состояние

FINCSTP	Увеличить указатель стека на 1
FDECSTP	Уменьшить указатель стека на 1
FFREE	Освободить регистр
FNOP	Холостая команда, нет операции
FSETPM	Установить защищенный режим работы

Команда FNSTCW записывает содержимое управляющего регистра в оперативную память.

Команда FLDCW загружает управляющий регистр данными из оперативной памяти и обычно используется для изменения режима работы сопроцессора.

Команда FNSTSW записывает содержимое регистра состояния в оперативную память. Команда FNSTSW AX записывает содержимое этого регистра в регистр AX центрального процессора для его последующего анализа командами условных переходов.

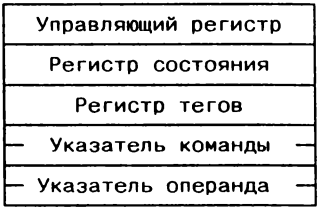
Сопроцессор 8087 не имеет варианта команды FSTSW AX, поэтому приходится вначале записывать регистр состояния в память, а затем в регистр флагов процессора 8086.

Команда FNCLEX сбрасывает флаги особых случаев в регистре состояния сопроцессора. Кроме того, сбрасываются биты ES и B.

Команда FNINIT инициализирует регистр состояния, управляющий регистр и регистр тегов в соответствии со следующей таблицей:

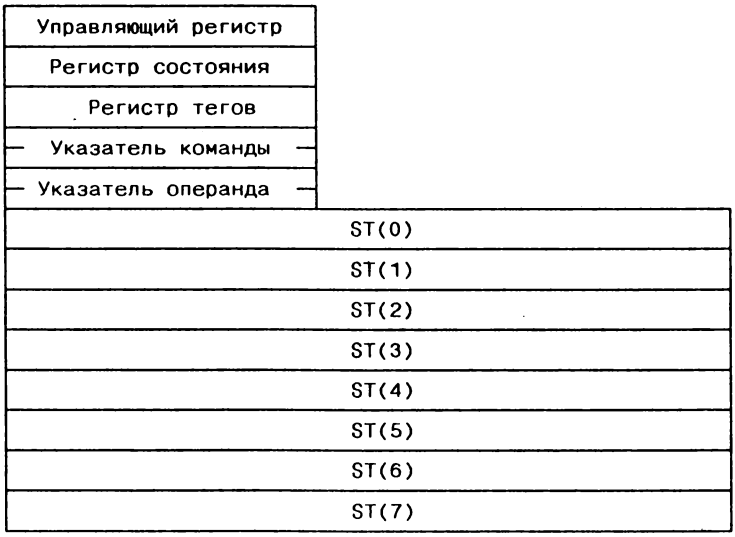
Регистр	Устанавливаемый режим работы
Управляющий	Проективная бесконечность, округление к ближайшему, расширенная точность, все особые случаи замаскированы
Состояния	B = 0 (бит занятости сброшен), код условия не определен, ST = ES = 0, флаги особых случаев установлены в нуль
Тегов	Все поля регистра тегов содержат значение 11 (пустой регистр)

Команда FNSTENV записывает в память содержимое всех регистров, кроме численных, в следующем формате:



Команда FLDENV предназначена для загрузки регистров, сохраненных ранее командой FNSTENV. Обе эти команды полезны в программах обработки особых случаев.

Команды FNSAVE и FRSTOR действуют аналогично командам FNSTENV и FLDENV, но дополнительно сохраняют и восстанавливают содержимое численных регистров. Формат области сохранения регистров, занимающей 94 байта, приведен на рисунке:



Команды FINCSTP и FDECSTP соответственно увеличивают и уменьшают на 1 указатель стека SP.

Команда FFREE ST(i) помечает численный регистр ST(i) как пустой, записывая в соответствующее поле регистра тегов значение 11.

Команда FNOP не производит никаких действий.

Команда FSETPM переводит сопроцессор в защищенный режим работы. Подробное рассмотрение защищенного режима работы выходит за рамки данной книги.

12.5. Программирование сопроцессора

Используя языки высокого уровня, такие, как Си или Паскаль, Вы можете даже и не знать, что созданная Вами программа использует для вычислений арифметический сопроцессор. При установке системы программирования QuickC или C 6.0 Вам предоставляется возможность выбора одного из трех вариантов стандартной библиотеки:

- библиотека эмулятора;
- библиотека, рассчитанная на наличие сопроцессора;
- библиотека альтернативной математики.

Первый вариант (библиотека эмулятора) используется по умолчанию. Программы, которые создаются с использованием эмулятора, будут работать как при наличии в системе сопроцессора, так и при его отсутствии. В последнем случае вычисления с плавающей точкой выполняются специальными подпрограммами, которые присоединяются к Вашей программе на этапе редактирования. Ваша программа сама определит факт наличия (или отсутствия) сопроцессора и выберет соответствующий способ выполнения вычислений - либо с использованием сопроцессора, либо с использованием подпрограмм эмуляции сопроцессора.

Все, что Вам нужно для работы с библиотекой эмуляции, - просто выбрать ее при установке системы программирования. Это самый простой способ программирования сопроцессора, когда Вам, вообще говоря, совсем не надо его программировать - всю работу по использованию сопроцессора выполнят модули библиотеки эмуляции.

Второй вариант библиотеки рассчитан на наличие сопроцессора. Если сопроцессора нет, программа работать не будет. Но если известно, что сопроцессор есть (например, процессор 80486 всегда содержит блок арифметики), то Вам имеет смысл использовать именно этот вариант, как самый быстродействующий.

Третий вариант не использует сопроцессор. Вычисления выполняются специальными подпрограммами, входящими в состав библиотеки альтернативной математики и подключающимися к Вашей программе автоматически на этапе редактирования.

К сожалению, есть программы, в которых использование библиотеки эмуляции невозможно или крайне затруднительно:

- резидентные программы;
- драйверы;
- программы, предъявляющие жесткие требования к точности и скорости вычислений.

В случае с резидентными программами невозможность использования библиотеки эмулятора вызвана тем, что после оставления резидентной программы в памяти, например функцией _dos_keep(), она теряет доступ к модулям эмуляции. Механизм вызова программ эмуляции основан на использовании прерываний с номерами 34h...3Eh. Перед тем как оставить программу резидентной, функция _dos_keep() восстанавливает содержимое этих векторов, делая невозможным доступ резидентной программе к модулям эмулятора. Да и самих этих модулей уже нет в памяти, на их место может быть загружена новая программа.

Поэтому руководство по Си рекомендует для резидентных программ использовать библиотеку альтернативной математики. Но эта библиотека, увы, не использует сопроцессор.

Ситуация с драйверами аналогична - драйверы, как правило, составляются на языке ассемблера, поэтому средства эмуляции библиотек Си недоступны.

Выходом может быть непосредственное программирование сопроцессора на языке ассемблера. При этом Вы можете полностью использовать все возможности сопроцессора и добиться от программы наибольшей эффективности вычислений.

Какие средства можно использовать для составления программ для сопроцессора?

Обычно это или ассемблер MASM (возможно использование TASM), либо интегрированная среда разработки QuickC версии 2.01, содержащая встроенный Quick Assembler.

Приведем пример самой простой программы, подготовленной для трансляции программой Quick Assemler. Эта программа выполняет вычисления по следующей несложной формуле:

```
z = x + y;
```

Значения x и y задаются в виде констант:

```
                .MODEL  TINY
                .STACK  100h
                .DATA
; Здесь находятся константы с одинарной точностью x и y
x   dd 1.0
y   dd 2.0
; Резервируем четыре байта для результата
z   dd ?
                .CODE
                .STARTUP
; Записываем в стек численных регистров значение x
                fld     x
; Складываем содержимое верхушки стека с константой y
                fadd    y
; Записываем результат в ячейку z
                fstp    z
; Завершаем работу программы и
; возвращаем управление операционной системе
                .EXIT   0
                END
```

Как убедиться в том, что программа работает правильно? Для этого мы используем отладчик CodeView, содержащий очень удобные средства отладки программ, работающих с арифметическим сопроцессором. Запустим отладчик CodeView, передав ему в качестве параметра имя приведенной выше программы:

```
cv test87.com
```

После того как отладчик запустится, откройте окно регистров сопроцессора, нажав комбинацию клавиш Alt-V-7 (см. верхний рисунок на стр. 120). В нижней части экрана появится окно регистров сопроцессора (см. нижний рисунок на стр. 120). Пусть Вас

не смущает то, что в этом окне пока не показывается состояние регистров сопроцессора. Нажмите клавишу F8, выполнив один шаг программы. Окно сопроцессора будет содержать информацию, вид которой показан на верхнем рисунке на стр. 121.

Теперь Вы видите содержимое регистров управления и состояния (cControl, cStatus), регистра тегов (cTag), регистров указателей команд и данных (Instr Ptr, Data Ptr), код выполняемой команды (Opcode). Отображается также содержимое стека численных регистров (Stack), но пока это поле пустое, т. к. все численные регистры отмечены в регистре тегов как пустые (код 11).

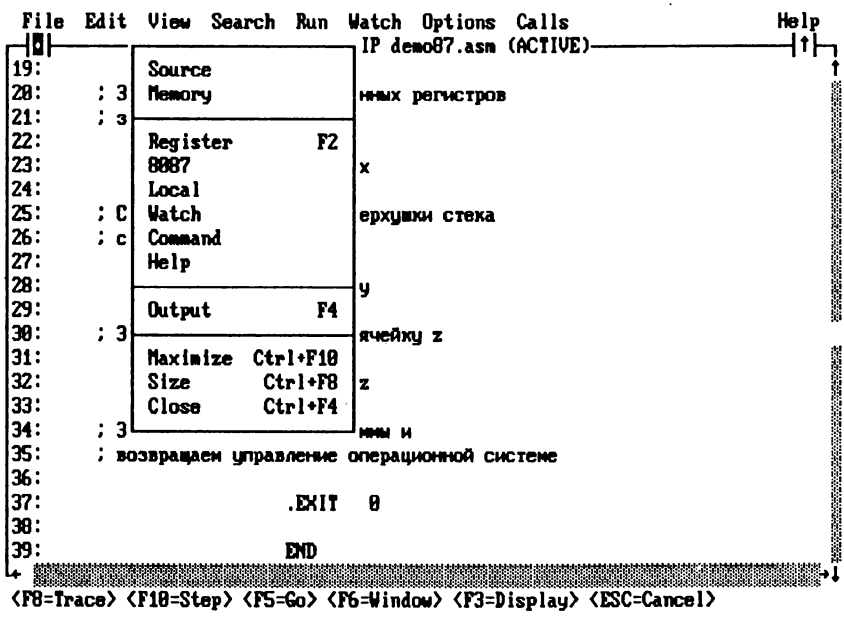
Нажмите еще раз клавишу F8, выполнив следующую команду программы. Эта команда запишет в стек численных регистров значение переменной x (см. нижний рисунок на стр. 121). Теперь в области регистров стека показано содержимое регистра cST(0), причем как в двоичном виде, так и с использованием экспоненциальной (научной) нотации. Как следовало ожидать, регистр ST(0) содержит величину 1.0.

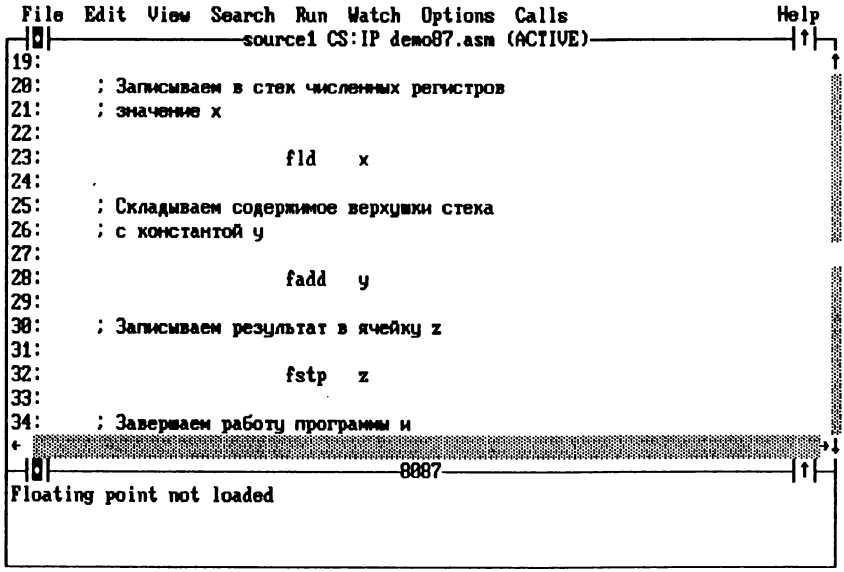
Выполним еще одну команду, прибавляющую к содержимому ST(0) значение 2.0 из переменной y. Теперь регистр ST(0) содержит величину 3.0 (см. верхний рисунок на стр. 122).

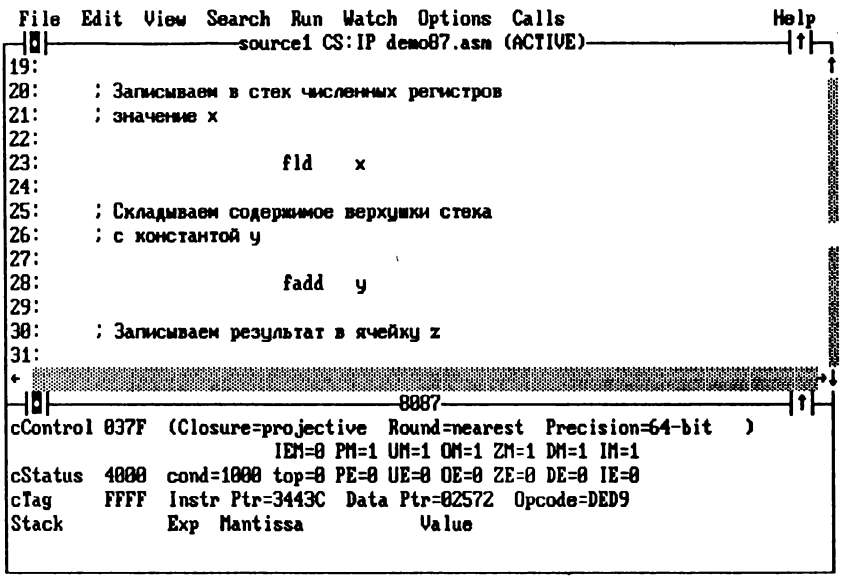
Последняя команда выталкивает из стека хранящееся там значение (3.0) и записывает его в переменную z. Теперь стек численных регистров снова пуст (см. нижний рисунок на стр. 122).

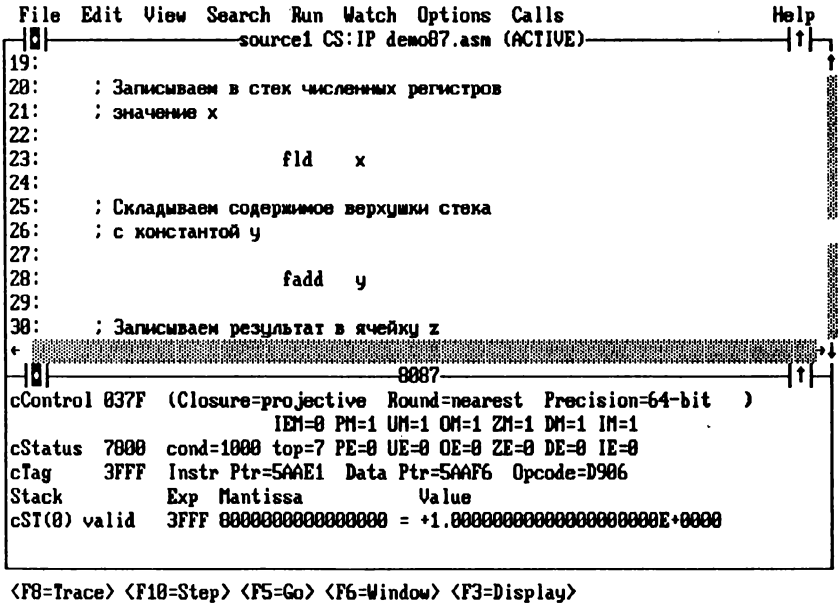
Отладчик CodeView обладает мощными средствами динамического просмотра состояния сопроцессора. Однако этот отладчик невозможно использовать для отладки драйверов. Мы уже говорили Вам о проблемах, возникающих при отладке драйверов, в первом томе "Библиотеки системного программиста".

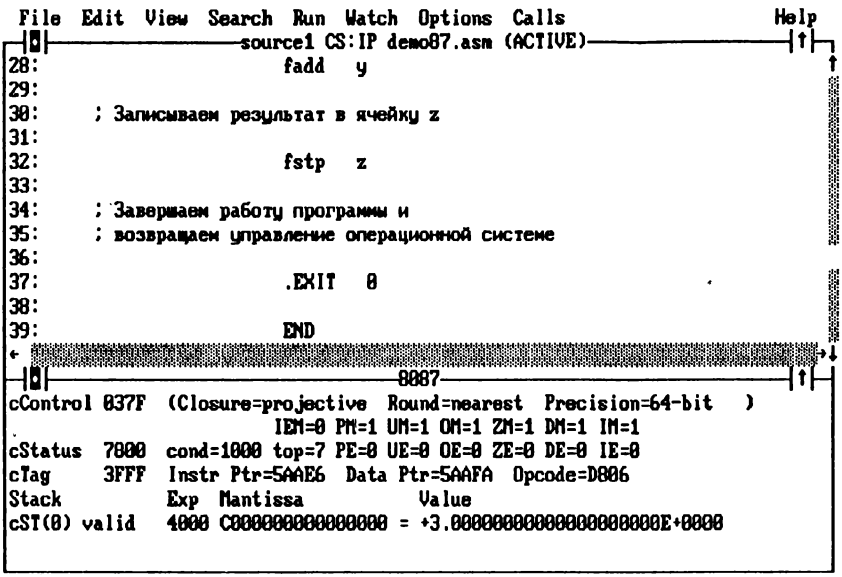
Там же нами была предложена методика отладки драйверов, основанная на включении в исходный текст драйвера подпрограмм, выводящих на экран содержимое регистров центрального процессора или областей памяти. Мы привели исходный текст подпрограммы ntrace, которая выводит на экран содержимое всех регистров центрального процессора.

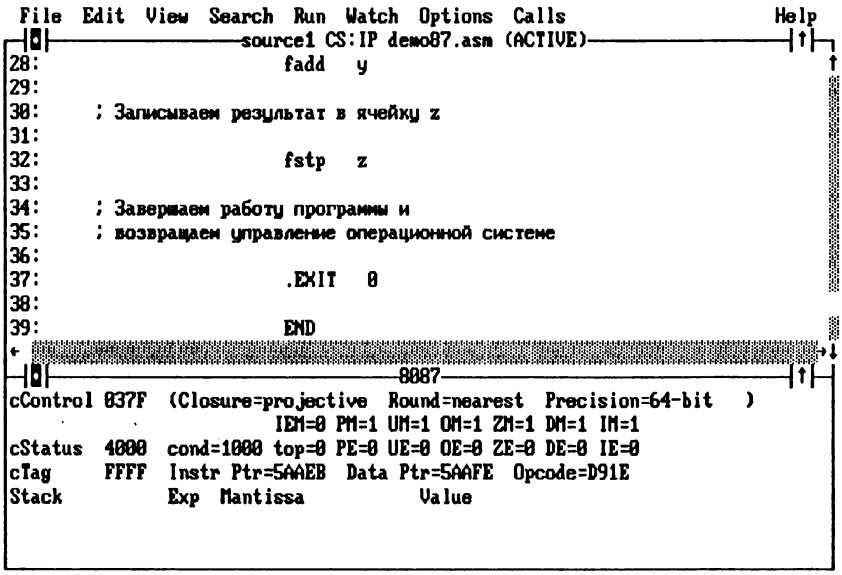

File Edit View Search Run Watch Options Calls Help
IP demo87.asm (ACTIVE)
Source
Memory
Register F2
8087
Local
Watch
Command
Help
Output F4
Maximize Ctrl+F10
Size Ctrl+F8
Close Ctrl+F4
; возвращаем управление операционной системе
.EXIT 0
END
<F8=Trace> <F10=Step> <F5=Go> <F6=Window> <F3=Display> <ESC=Cancel>


File Edit View Search Run Watch Options Calls Help
source1 CS:IP demo87.asm (ACTIVE)
; Записываем в стек численных регистров
; значение x
fld x
; Складываем содержимое верхушки стека
; с константой y
fadd y
; Записываем результат в ячейку z
fstp z
; Завершаем работу программы и
8087
Floating point not loaded
<F8=Trace> <F10=Step> <F5=Go> <F6=Window> <F3=Display>


File Edit View Search Run Watch Options Calls Help
source1 CS:IP demo87.asm (ACTIVE)
19:
20: ; Записываем в стек численных регистров
21: ; значение x
22:
23: fld x
24:
25: ; Складываем содержимое верхушки стека
26: ; с константой y
27:
28: fadd y
29:
30: ; Записываем результат в ячейку z
31:
8087
cControl 037F (Closure=projective Round=nearest Precision=64-bit)
IEM=0 PM=1 UM=1 OM=1 ZM=1 DM=1 IM=1
cStatus 4000 cond=1000 top=0 PE=0 UE=0 OE=0 ZE=0 DE=0 IE=0
cTag FFFF Instr Ptr=3443C Data Ptr=02572 Opcode=DED9
Stack Exp Mantissa Value
<F8=Trace> <F10=Step> <F5=Go> <F6=Window> <F3=Display>


File Edit View Search Run Watch Options Calls Help
source1 CS:IP demo87.asm (ACTIVE)
19:
20: ; Записываем в стек численных регистров
21: ; значение x
22:
23: fld x
24:
25: ; Складываем содержимое верхушки стека
26: ; с константой y
27:
28: fadd y
29:
30: ; Записываем результат в ячейку z
8087
cControl 037F (Closure=projective Round=nearest Precision=64-bit)
IEM=0 PM=1 UM=1 OM=1 ZM=1 DM=1 IM=1
cStatus 7000 cond=1000 top=7 PE=0 UE=0 OE=0 ZE=0 DE=0 IE=0
cTag 3FFF Instr Ptr=5AAE1 Data Ptr=5AAF6 Opcode=D906
Stack Exp Mantissa Value
cST(0) valid 3FFF 8000000000000000 = +1.0000000000000000000E+0000
<F8=Trace> <F10=Step> <F5=Go> <F6=Window> <F3=Display>


File Edit View Search Run Watch Options Calls Help
source1 CS:IP demo87.asm (ACTIVE)
28: fadd y
29:
30: ; Записываем результат в ячейку z
31:
32: fstp z
33:
34: ; Завершаем работу программы и
35: ; возвращаем управление операционной системе
36:
37: .EXIT 0
38:
39: END
8087
cControl 037F (Closure=projective Round=nearest Precision=64-bit)
IEM=0 PM=1 UM=1 OM=1 ZM=1 DM=1 IM=1
cStatus 7800 cond=1000 top=7 PE=0 UE=0 OE=0 ZE=0 DE=0 IE=0
cTag 3FFF Instr Ptr=5AAE6 Data Ptr=5AAFA Opcode=D806
Stack Exp Mantissa Value
cST(0) valid 4000 C000000000000000 = +3.0000000000000000000E+0000
<F8=Trace> <F10=Step> <F5=Go> <F6=Window> <F3=Display>


File Edit View Search Run Watch Options Calls Help
source1 CS:IP demo87.asm (ACTIVE)
28: fadd y
29:
30: ; Записываем результат в ячейку z
31:
32: fstp z
33:
34: ; Завершаем работу программы и
35: ; возвращаем управление операционной системе
36:
37: .EXIT 0
38:
39: END
8087
cControl 037F (Closure=projective Round=nearest Precision=64-bit)
IEM=0 PM=1 UM=1 OM=1 ZM=1 DM=1 IM=1
cStatus 4000 cond=1000 top=0 PE=0 UE=0 OE=0 ZE=0 DE=0 IE=0
cTag FFFF Instr Ptr=5AAEB Data Ptr=5AAFE Opcode=D91E
Stack Exp Mantissa Value
<F8=Trace> <F10=Step> <F5=Go> <F6=Window> <F3=Display>

Если Ваш драйвер использует сопроцессор, Вам, вероятно, потребуется также содержимое регистров сопроцессора. Приведем текст подпрограммы ntrace87, которая наряду с содержимым регистров центрального процессора выводит содержимое регистров арифметического сопроцессора:

```
      include sysp.inc
     .MODEL tiny
     .CODE
     PUBLIC ntrace87
;=============================================================
; Процедура выводит на экран содержимое всех регистров
; центрального процессора и сопроцессора. Затем она ожидает
; нажатия на любую клавишу. После возвращения из процедуры
; все регистры восстанавливаются, в том числе регистры
; сопроцессора.
ntrace87 proc near
; Сохраняем в стеке регистры,
; содержимое которых будет изменяться
     pushf
     push ax
     push bx
     push cx
     push dx
     push ds
     push bp
     mov bp,sp
       push cs
       pop ds
; Сохраняем полное состояние сопроцессора
       fsave  cs:regs_87
; Выводим сообщение об останове
       mov dx,offset cs:trace_msg
       @@out_str
; Выводим содержимое всех регистров
       mov ax,cs          ; cs
     call Print_word
     @@out_ch ':'
     mov ax,[bp]+14     ; ip
     call Print_word
```

```
    @@out_ch 13,10,13,10,'A','X','='
    mov ax,[bp]+10
    call Print_word

    @@out_ch ' ','B','X','='
    mov ax,[bp]+8
    call Print_word

    @@out_ch ' ','C','X','='
    mov ax,[bp]+6
    call Print_word

    @@out_ch ' ','D','X','='
    mov ax,[bp]+4
    call Print_word

    @@out_ch ' ','S','P','='
    mov ax,bp
    add ax,16
    call Print_word

    @@out_ch ' ','B','P','='
    mov ax,[bp]
    call Print_word

    @@out_ch ' ','S','I','='
    mov ax,si
    call Print_word

    @@out_ch ' ','D','I','='
    mov ax,di
    call Print_word

    @@out_ch 13,10,'D','S','='
    mov ax,[bp]+2
    call Print_word

    @@out_ch ' ','E','S','='
    mov ax,es
    call Print_word

    @@out_ch ' ','S','S','='
    mov ax,ss
    call Print_word

    @@out_ch ' ','F','='
    mov ax,[bp]+12
    call Print_word

; Выводим содержимое регистров сопроцессора

    lea dx,cs:r87_msg
    @@out_str

; Выводим содержимое управляющего регистра

    @@out_ch 'C','N','T','R','='
```

```
    mov ax, cs:regs_87.cr
    call Print_word
; Выводим содержимое регистра состояния
    @@out_ch ' ','S','T','A','T','E','='
    mov ax, cs:regs_87.sr
    call Print_word
; Выводим содержимое регистра тегов
    @@out_ch ' ','T','A','G','='
    mov ax, cs:regs_87.tg
    call Print_word
; Выводим содержимое указателя адреса
    @@out_ch ' ','C','M','D','A','D','R','='
    mov ax, cs:regs_87.cmdhi
    and  ah, 0f0h
    mov al, ah
    mov cl, 4
    ror al, cl
    call Print_byte
    mov ax, cs:regs_87.cmdlo
    call Print_word
    @@out_ch ' '
; Выводим содержимое указателя операнда
    @@out_ch ' ','O','P','R','A','D','R','='
    mov ax, cs:regs_87.oprhi
    and  ah, 0f0h
    mov al, ah
    mov cl, 4
    ror al, cl
    call Print_byte
    mov ax, cs:regs_87.oprlo
    call Print_word
; Выводим содержимое непустых численных регистров
    lea dx,cs:nr_msg
    @@out_str
    mov cx, 8               ; количество регистров - 8
    mov dx, 0               ; индекс текущего регистра
    mov bx, cs:regs_87.tg   ; содержимое регистра тегов
; Цикл по стеку численных регистров
nreg_loop:
; Проверяем поле регистра тегов, соответствующее
; текущему обрабатываемому численному регистру
```

```
        mov ax, bx
        and ax, 0c000h
        cmp ax, 0c000h
; Если это поле равно 11B, считаем, что данный
; численный регистр пуст; переходим к следующему
        je continue
; Выводим на экран содержимое численного регистра
        call Print_numreg
continue:
; Сдвигаем содержимое регистра тегов для обработки поля,
; соответствующего следующему регистру.
        rol bx, 1
        rol bx, 1
        inc dx          ; увеличиваем индекс текущего регистра
        loop nreg_loop
        lea dx,cs:hit_msg
        @@out_str
; Ожидаем нажатия на любую клавишу
      mov ax,0
      int 16h
; Восстанавливаем содержимое регистров.
      frstor  cs:regs_87
      pop bp
      pop ds
      pop dx
      pop cx
      pop bx
      pop ax
      popf
      ret
trace_msg db 13,10,'>---- BREAK ----> At address ','$'
hit_msg db 13,10,'Hit any key...','$'
r87_msg db 13,10,13,10,'Coprocessor state:',13,10,'$'
nr_msg db 13,10,'Numeric Registers:',13,10,'$'
regs_87 db 94 dup(?)
ten db 10
ntrace87 endp
;=============================================================
; Процедура выводит на экран содержимое численного регистра
; с номером, заданным в регистре al
```

```
Print_numreg proc near
        push cx
        push bx

; Выводим обозначение численного регистра

        push dx
        @@out_ch 'S','T','('
        pop dx
        mov al, dl
        call Print_byte
        push dx
        @@out_ch ')','='
        pop dx

; Выводим содержимое численного регистра в
; шестнадцатеричном формате

        mov cx, 10      ; счетчик байтов в числе с
                                ; расширенной точностью
        mov bp, 10      ; первоначальное смещение
                                ; к старшему байту числа

; Смещение к полю первого численного регистра
; в области сохранения

        mov bx, offset cs:regs_87.st0

; Вычисляем смещение старшего байта численного
; регистра, номер которого задан в регистре DX

        mov ax, dx
        imul cs:ten
        add bx, ax
        dec bx

; Выводим в цикле 10 байтов числа

pr_lp:
        push bx

        add bx, bp
        mov al, cs:[bx]
        call Print_byte
        pop bx

        dec bp
        loop pr_lp

        push dx
        @@out_ch 13,10
        pop dx

        pop bx
        pop cx
        ret

Print_numreg endp
```

```
;=========================================
; Процедура выводит на экран содержимое AL

Print_byte proc near

       push ax
       push bx
       push dx

       call Byte_to_hex
       mov bx,dx
       @@out_ch bh
       @@out_ch bl

       pop dx
       pop bx
       pop ax
       ret
Print_byte endp

;=========================================
; Процедура выводит на экран содержимое AX

Print_word proc near

     push ax
     push bx
     push dx

     push ax
     mov cl,8
     rol ax,cl
     call Byte_to_hex
     mov bx,dx
     @@out_ch bh
     @@out_ch bl

     pop ax
     call Byte_to_hex
     mov bx,dx
     @@out_ch bh
     @@out_ch bl

     pop dx
     pop bx
     pop ax
     ret
Print_word endp

Byte_to_hex proc near
;--------------------
; al - input byte
; dx - output hex
;--------------------
     push ds
```

```
        push cx
        push bx

        lea bx,tabl
        mov dx,cs
        mov ds,dx

        push ax
        and al,0fh
        xlat
        mov dl,al

        pop ax
        mov cl,4
        shr al,cl
        xlat
        mov dh,al

        pop bx
        pop cx
        pop ds
        ret
tabl db '0123456789ABCDEF'
Byte_to_hex endp

        end
```

Работа программы основана на использовании команды FSAVE, сохраняющей в памяти содержимое всех регистров сопроцессора. Область сохранения описывается следующей структурой, определенной в файле sysp.inc:

```
State87 struc
        cr dw ?
        sr dw ?
        tg dw ?
        cmdlo dw ?
        cmdhi dw ?
        oprlo dw ?
        oprhi dw ?
        st0 dt ?
        st1 dt ?
        st2 dt ?
        st3 dt ?
        st4 dt ?
        st5 dt ?
        st6 dt ?
        st7 dt ?
State87 ends
```

Для демонстрации возможностей ntrace87 мы немного изменили нашу первую программу, работающую с сопроцессором - после каждой команды сопроцессора вставили вызов ntrace87:

```
            .MODEL  tiny
            DOSSEG

            EXTRN ntrace87:NEAR

            .STACK  100h

            .DATA
x   dd 1.0
y   dd 2.0
; Резервируем четыре байта для результата
z   dd ?

            .CODE
            .STARTUP

            push cs
            pop  ds

; Записываем в стек численных регистров значение x
            call ntrace87

            fld    x
            call ntrace87

; Складываем содержимое верхушки стека с константой y

            fadd   y
            call ntrace87

; Записываем результат в ячейку z

            fstp   z
            call ntrace87

; Завершаем работу программы и
; возвращаем управление операционной системе

quit:
            .EXIT   0

            END
```

В процессе работы этой программы на каждом шаге на экран выводится дамп содержимого регистров центрального процессора и сопроцессора (пустые численные регистры не отображаются):

```
>---- BREAK ----> At address 2314:0105

AX=0000 BX=0000 CX=00FF DX=2314 SP=FFFE BP=091C SI=0100 DI=FFFE
DS=2314 ES=2314 SS=2314 F=7202
```

```
Coprocessor state:
CNTR=037F STATE=4000 TAG=FFFF CMDADR=023256  OPRADR=02365E
Numeric Registers:

Hit any key...
>---- BREAK ----> At address 2314:010D

AX=0000 BX=0000 CX=00FF DX=2314 SP=FFFE BP=091C SI=0100 DI=FFFE
DS=2314 ES=2314 SS=2314 F=7202

Coprocessor state:
CNTR=037F STATE=7800 TAG=3FFF CMDADR=023246  OPRADR=02365E
Numeric Registers:
ST(00)=3FFF8000000000000000

Hit any key...
>---- BREAK ----> At address 2314:0115

AX=0000 BX=0000 CX=00FF DX=2314 SP=FFFE BP=091C SI=0100 DI=FFFE
DS=2314 ES=2314 SS=2314 F=7202

Coprocessor state:
CNTR=037F STATE=7800 TAG=3FFF CMDADR=02324E  OPRADR=023662
Numeric Registers:
ST(00)=4000C000000000000000

Hit any key...
>---- BREAK ----> At address 2314:011D

AX=0000 BX=0000 CX=00FF DX=2314 SP=FFFE BP=091C SI=0100 DI=FFFE
DS=2314 ES=2314 SS=2314 F=7202

Coprocessor state:
CNTR=037F STATE=4000 TAG=FFFF CMDADR=023256  OPRADR=023666
Numeric Registers:

Hit any key...
```

12.6. Обработка особых случаев

В арифметическом сопроцессоре имеются два механизма обработки ошибок, возникающих при выполнении различных команд. Первый механизм основан на генерации так называемого прерывания особого случая (INT 10h). Это прерывание вырабатывается в том случае, когда происходит какая-нибудь ошибка (например, деление на нуль) при условии, что соответствующие биты масок особых случаев в регистре управления не установлены. При втором способе обработки ошибок все особые случаи маскируются (соответствующие биты управляющего регистра устанавливаются в единицу) и в случае ошибки сопроцессор в качестве результата возвращает некоторое заранее известное особое значение (нечис-

ло, неопределенность или бесконечность). Программист может выбирать между этими способами обработки ошибок, маскируя или разрешая прерывание по особому случаю. Если прерывание особого случая замаскировано, можно предложить следующий способ обнаружения ошибки:

- сбросить флажки особых случаев в регистре сосотояния;
- выполнить одну или несколько команд сопроцессора;
- проверить состояние флажков особых случаев в регистре состояния, в частности бит суммарной ошибки ES;
- если какой-либо флажок установлен, вызвать программу обработки ошибочной ситуации;
- в программе обработки ошибочной ситуации можно сбросить флажки особых случаев, записав соответствующее значение в регистр состояния.

Кроме того, после выполнения команды полезно проверить получившийся результат на принадлежность к множеству особых значений.

Рассмотрим возможные особые случаи сопроцессора в реальном режиме.

12.6.1. Неточный результат

В результате выполнения некоторых операций может возникнуть такая ситуация, когда невозможно точно представить результат. Например, результатом деления числа 1.0 на 3.0 является бесконечная периодическая двоичная дробь 0.010101... Такое число не может быть представлено точно ни в одном формате вещественных чисел.

Обычно неточный результат является результатом округления и может не рассматриваться как ошибка.

12.6.2. Переполнение

Если результат выполнения операции слишком велик и не может быть представлен в формате приемника результата, фиксируется особый случай переполнения.

Этот особый случай обязательно произойдет, например, при сложении максимального числа расширенной точности с самим

собой или при преобразовании этого числа в формат с двойной или одинарной точностью.

Так как для хранения промежуточных результатов используется 80-битовое представление, при выполнении операций над числами с одинарной или двойной точностью переполнения, как правило, не происходит. Огромный диапазон чисел с расширенной точностью гарантирует правильность представления больших по абсолютной величине результатов операций с числами одинарной и двойной точности.

12.6.3. Антипереполнение

Антипереполнение возникает тогда, когда результат слишком мал для его представления в формате приемника результата операции, но все же отличен от нуля. Например, если делается попытка преобразовать наименьшее положительное число с расширенной точностью в формат числа с двойной или одинарной точностью.

Если Вы используете числа только с двойной или одинарной точностью, а для хранения промежуточных результатов используете формат с расширенной точностью, особый случай антипереполнения, как правило, не возникает.

12.6.4. Деление на нуль

Этот особый случай возникает при попытке выполнить деление конечного ненулевого числа на нуль.

В афинном режиме при делении конечных (положительных или отрицательных) чисел на нуль (положительный или отрицательный) в качестве результата возвращается бесконечность. Знак этой бесконечности зависит от знака делимого и от знака нуля. Например, при делении положительного ненулевого числа на положительный нуль получается положительная бесконечность, при делении положительного ненулевого числа на отрицательный нуль - отрицательная бесконечность.

В проективном режиме, а также при попытке деления нуля на нуль возникает особый случай недействительной операции, который будет рассмотрен ниже.

12.6.5. Недействительная операция

Этот особый случай возникает при попытке выполнения таких запрещенных команд, как деление нуля на нуль, извлечения корня из отрицательного числа, обращение к несуществующему регистру сопроцессора или при попытке использования в качестве операндов команд нечисел, неопределенностей или бесконечности (для трансцендентных функций).

12.6.6. Денормализованный операнд

Мы уже говорили о том, что сопроцессор использует операнды в нормализованной форме. Однако при выполнении операции может оказаться, что результат слишком мал по абсолютной величине для представления его в нормализованной форме. Можно было бы считать такой результат нулевым, однако это привело бы к снижению точности вычислений или даже к грубым ошибкам. Например, вычисляется следующее выражение:

```
(y-x)+x;
```

Если разность (y - x) вызывает антипереполнение и в качестве результата берется нулевое значение, то после вычисления всего выражения получится x. Если же пойти на расширение диапазона представления чисел за счет снижения точности и сформировать результат вычисления разности (y-x) как денормализованное число, выражение будет вычислено правильно и в результате получится y.

Таким образом, иногда целесообразно замаскировать особый случай денормализованного операнда и использовать денормализованные числа. Однако при попытке деления на ненормализованное число или извлечения из него квадратного корня фиксируется особый случай недействительной операции.

ОБЗОР ЛИТЕРАТУРЫ

Undocumented DOS
Andrew Schulman, Raymond J. Michels, Jim Kyle, Tim Paterson, David Maxey, Ralf Brown

ISBN 0-201-57064-5
QA76.76.O63U53 1990
005.4'46-dc20 90-46992
Second Printing, February 1991. - 679 c.

Книга посвящена использованию недокументированных прерываний MS-DOS версии до 4.01 включительно. В книге разъясняются причины отсутствия в документации по операционной системе информации о некоторых прерываниях и структурах данных. Кроме того, показана важность этих прерываний. Программы могут вполне безопасно использовать некоторые недокументированные особенности операционной системы без риска потери совместимости со следующими версиями MS-DOS.

В книге рассмотрен механизм управления памятью в MS-DOS, процесс загрузки и инициализации драйверов.

Описана внутренняя структура файловой системы, множество недокументированных прерываний и структур данных, связанных с файловой системой. Описаны особенности работы с файловой системой в сети.

Отдельная глава посвящена составлению резидентных программ. Показано, что без использования недокументированных прерываний невозможно составить правильно работающую резидентную программу

Описана работа командного интерпретатора MS-DOS, приведена информация, необходимая для разработки собственного интерпретатора.

Описаны прерывания, необходимые для разработки отладчиков программ, таких, как Code View и DEBUG. Большой раздел посвящен отладке программ в среде WINDOWS.

К книге прилагаются дискеты, содержащие исходные тексты всех программ и справочную базу данных. В справочной базе данных собрана информация о недокументированных прерываниях и структурах данных. В частности, описан интерфейс с WINDOWS с различным сетевым программным обеспечением.

Advanced DOS. Memory resident utilities, interrupts, and disk managment with MS- and PC-DOS
Michael I. Hyman

ISBN 0-943518-83-0
Second Edition
MIS INC, 1988

Эта книга содержит достаточно подробное описание логической структуры диска в MS-DOS (версии 3.30 и более ранних версий).

В первой части книги описана работа с файлами и каталогами на уровне прерываний MS-DOS, структура таблицы разделов диска.

Вторая часть книги посвящена использованию прерываний в программах, составленных на языке ассемблера. Отдельная глава посвящена виде-

осистеме компьютера (будет рассмотрена нами в следующем томе "Библиотеки системного программиста"). В этой части книги приведена информация об использовании клавиатуры, мыши и светового пера, подробные сведения о файловой системе MS-DOS. Вводится понятие расширенной и дополнительной памяти. Для дополнительной памяти описываются несколько функций прерывания INT 67h, обслуживающего запросы к этой памяти.

Третья часть книги содержит информацию, полезную при составлении резидентных программ.

Книга содержит программы на языках ассемблера и Паскаль.

Superchargin Your PC
Lewis Perdue

ISBN 0-07-881000-0
Brkley, California 94710

Книга посвящена модернизации аппаратного обеспечения персонального компьютера. Описано подключение разнообразных периферийных устройств, таких, как дополнительные дисководы, модемы, и т. д. Описаны процедура подключения дополнительной оперативной памяти, принципы объединения компьютеров в сети. Приводятся сведения об использовании параллельного принтерного порта и последовательного порта RS-232S. Многочисленные рисунки и фотографии облегчают работу с книгой.

Микропроцессоры и микропроцессорные комплекты интегральных микросхем. Том 2
Под ред. В. А. Шаханова

ISBN 5-256-00373-9
М.: Радио и связь, 1988

В этом справочнике описано несколько микросхем серии К1810, в частности контроллер прерываний КР1810ВН59А, совместимый с используемыми в персональных компьютерах IBM PC/XT/AT контроллерами 8259А.

Приведены форматы команд и внутренних регистров контроллера, описано функционирование контроллера в различных режимах.

Микропроцессорный комплект К1810
Под редакцией Ю. М. Казаринова.

ISBN 5-06-000821-5
М.: Высшая школа, 1990

В книге приведены справочные данные на микросхемы серии К1810. Наибольшую ценность для программиста представляет описание микросхем центрального процессора К1810ВМ86 (аналог Intel 8086), арифметического сопроцессора К1810ВМ87 (аналог Intel 8087), интервального таймера К1810ВИ54 (аналог таймера Intel 8254), контроллера прямого доступа к памяти К1810ВТ37 (аналог Intel 8237). Книгу можно рекомендовать системным программистам в качестве справочника по программированию перечисленных выше микросхем.

Архитектура микропроцессора 80286
С. П. Морс, Д. Д. Алберт

М.: Радио и связь, 1990

В книге подробно описаны архитектура и система команд процессора Intel 80286, арифметического сопроцессора Intel 80287. Рассматриваются реальный и защищкнный режимы работы процессора и сопроцессора, большое внимание уделяется организациии работы мультизадачного программного обеспечения.

Введение в схемотехнику ПЭВМ IBM PC/AT
Левкин Г. Н., Левкина В. Е.

ISBN 5-7043-0562-8
М.: МГПИ, 1991

Описывается схемотехника первоначального варианта компьютера IBM AT. Книга содержит описание микросхем 80286 (центральный процессор), 82284 (тактовый генератор), 82288 (шинный контроллер). Описание ориентировано на разработчиков аппаратных средств, т. к. в книге ничего не сказано о программировании описанных микросхем.

Приведена принципиальная схема материнской платы компьютера IBM AT и часть листинга BIOS, имеющая отношение к самопроверке компьютера при включении питания и сбросе. Листинг снабжен подробными комментариями и позволяет проследить процесс инициализации отдельных подсистем компьютера.

Персональные ЭВМ IBM PC и XT
Л. Скэнлон

ISBN 5-256-00956-7
М.: Радио и связь, 1991

В книге рассмотрены вопросы программирования на языке ассемблера для персональных компьютеров IBM PC/XT. Приводятся сведения как по языку ассемблера, так и по аппаратному обеспечению персонального компьютера. Книга доступна для тех, кто только начинает использовать язык ассемблера. К сожалению, отсутствуют сведения по программированию широко распространенного компьютера IBM AT - следующей модели персональных компьютеров после IBM XT.

Приложение 1

КОДЫ КЛАВИАТУРЫ

Таблица скан-кодов для клавиатуры IBM PC/XT

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

45	NumLock						
46	ScrollLock						
47	Home [7]						
48	Up [8]						
49	PgUp [9]						
4a	K -						
4b	<- [4]						
4c	[5]						
4d	-> [6]						
4e	K +						
4f	End [1]						
50	Dn [2]						
51	PgDn [3]						
52	Ins [0]						
53	Del [.]						

Таблица расширенного ASCII-кода

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	NUL	03	^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

Таблица ASCII-кодов клавиш, имеющихся только на 101-клавишной клавиатуре

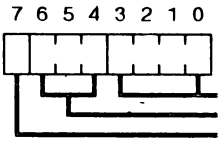
F11	85	Alt-Bksp	0e	Alt- Д /	a4
F12	86	Alt-Enter	1c	Alt- Д *	37
Shft-F11	87	Alt-Esc	01	Alt- Д -	4a
Shft-F12	88	Alt-Tab	a5	Alt- Д +	4e
Ctrl-F11	89	Ctrl-Tab	94	Alt- Д Enter	a6
Ctrl-F12	8a				
Alt-F11	8b	Alt-up Up	98	Ctrl- Д /	95
Alt-F12	8c	Alt-down Dn	a0	Ctrl- Д *	96
Alt-[	1a	Alt-left <-	9b	Ctrl- Д -	8e
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

Приложение 2

РЕГИСТРЫ ЧАСОВ РЕАЛЬНОГО ВРЕМЕНИ

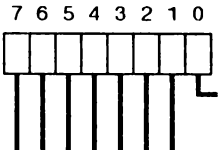
Регистр	*Назначение*
0	Счетчик секунд
1	Регистр секунд будильника
2	Счетчик минут
3	Регистр минут будильника
4	Счетчик часов
5	Регистр часов будильника
6	Счетчик дней недели (1 - воскресенье)
7	Счетчик дней месяца
8	Счетчик месяцев
9	Счетчик лет (последние две цифры текущего года)

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



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. (уст. в 0)
Разрешение прерывания по окончании изменения данных (установлен в 0)
Разрешение прерывания будильника (уст. 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.

Приложение 3

ЧАСТОТЫ НОТ ДЛЯ ВТОРОЙ ОКТАВЫ

В этой таблице приведены частоты нот для второй октавы.

При повышении (понижении) тона на октаву частота соответствующей ноты умножается (делится) на два.

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

Приложение 4

РАЗВОДКА РАЗЪЕМА ПОСЛЕДОВАТЕЛЬНОГО ПОРТА

Приведем разводку разъема порта 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 используется	
20	Готовность выходных данных (Data Terminal Ready, DTR)	Выход
21	Не используется	
22	Индикатор вызова (Ring Indicator, RI)	Вход
23 - 25	Не используется	

Приведем также разводку разъема порта последовательной передачи данных DB9P

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)	-"-

Приложение 5

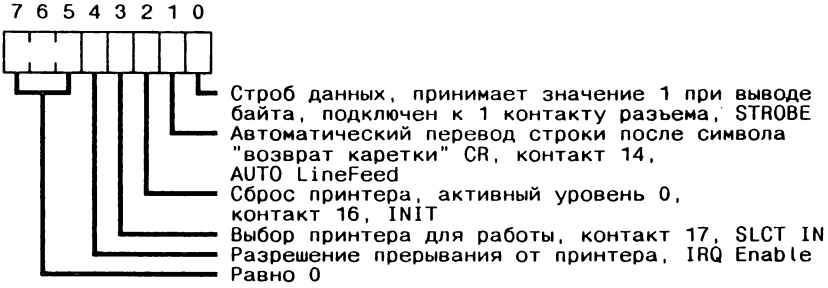
ПОРТЫ АДАПТЕРА ПРИНТЕРА

Порт 378h

Порт предназначен для записи выводимого на принтер байта данных. Возможно также чтение только что записанного байта.

Порт 37Ah

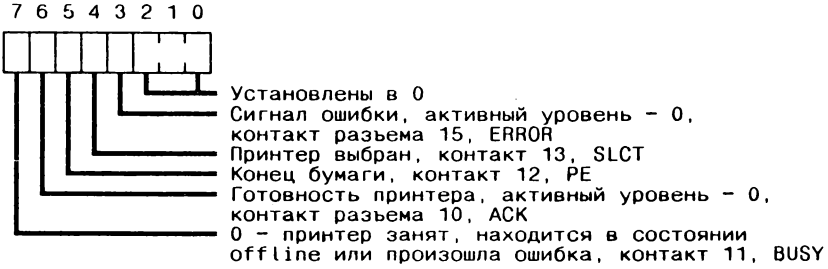
Порт управления принтером, доступен для чтения и записи:



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:



Приложение 6

РАЗВОДКА РАЗЪЕМОВ ПРИНТЕРНОГО ПОРТА

Таблица 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	-"- 1	-"-
4	4	-"- 2	-"-
5	5	-"- 3	-"-
6	6	-"- 4	-"-
7	7	-"- 5	-"-
8	8	-"- 6	-"-
9	9	-"- 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	-

Приложение 7

КОМАНДЫ ПРИНТЕРА Epson LQ-2550

Управление принтером

ESC @ - Инициализация принтера
1Bh 40h
Выполняется сброс принтера в исходное состояние, если происходила печать строки, то при инициализации эта печать отменяется.

DC1(17) - Выбор принтера для работы
11h
Возвращает принтер в выбранное (активное) состояние, если раньше он был переведен в неактивное состояние командой DC3. Эта команда не выполняет никаких действий, если принтер находится в состоянии OFF LINE.

DC3(19) - Перевод принтера в неактивное состояние
13h
Переводит принтер в неактивное состояние до получения принтером команды DC1 выбора принтера для работы. Если принтер переведен в неактивное состояние, его нельзя перевести в состояние ON LINE при помощи соответствующей клавиши на лицевой панели.

DEL(127) - Удаление символа
7Fh
Удаляет последний переданный в принтер символ. Эта команда не действует на управляющие коды.

ESC < - Печать в одном направлении
1Bh 3Ch
Переключает принтер в режим печати в одном направлении, т. е. принтер печатает только тогда, когда печатающая головка движется в одну сторону. Команда задает режим только для одной строки, символ возврата каретки отменяет печать только в одном направлении.

ESC U n - Включение/выключение режима печати в одном направлении
1Bh 55h n
В зависимости от параметра n режим печати только в одном направлении может включаться либо отключаться:

n = 1 режим включен;
n = 0 режим выключен.

ESC EM n - Управление податчиком бумаги
1Bh 19h n
Эта команда распознается принтером только в том случае, если принтер оборудован податчиком бумаги (Cut Sheet Feeder). Возможные значения параметра n:

1 Загрузка бумаги из кармана BIN1
2 Загрузка бумаги из кармана BIN2
R Проброс листа бумаги без загрузки нового.

BEL(7) - Выдача звукового сигнала
07h
Если выдать эту команду на принтер, громкоговоритель принтера издаст звуковой сигнал. Эта команда может быть использована для предупреждения оператора, например, о том, что кончилась бумага.

Управление старшим битом данных

Старший бит данных MSB (Most Significant Bit) - это бит 7 байтов, посылаемых программой в принтер. Обычно Вам не надо изменять значение старшего бита данных, более того, команды управления старшим битом не действуют в графическом режиме или при переопределении символов.

ESC = - Установка MSB в 0
1Bh 3Dh
Для всех выводимых на принтер данных старший бит принудительно устанавливается в 0. Этот режим используется при работе принтера с такими компьютерами, которые всегда посылают на принтер данные со старшим битом, установленным в 1.

ESC > - Установка MSB в 1
1Bh 3Eh
Для всех выводимых на принтер данных старший бит принудительно устанавливается в 1.

ESC # - Отмена управления MSB
1Bh 23h
Отменяется действие введенных ранее команд управления MSB "ESC =" или "ESC >".

Управление печатающей головкой и перемещением бумаги

CR(13) - Возврат каретки
0Dh
Команда вызывает распечатку содержимого буфера данных принтера, после чего головка переводится в начало текущей строки.

CAN(24) - Отмена печати строки
18h
Текст из распечатываемой строки удаляется. Команда не влияет на управляющие коды, находящиеся в этой строке.

FF(12) - Подача бумаги на один лист
0Ch
Команда вызывает распечатку содержимого буфера принтера, вслед за этим происходит подача бумаги вперед на один лист (в соответствии с установленной длиной листа).

ESC C n - Установить длину листа бумаги в строках
1Bh 43h n
Команда устанавливает длину листа бумаги равной n строкам. Используется установленное ранее расстояние между строками. Допустимые значения параметра n лежат в пределах 1...127 строк.

ESC C 0 n - Установить длину листа бумаги в дюймах
1Bh 43h 00h n
Команда устанавливает длину листа бумаги равной n дюймам. Значение n находиться в пределах 1...22 дюймов.

ESC N n - Установить режим пропуска перфорации
1Bh 4Eh n
В этой команде параметр n - это количество строк, пропускаемых принтером между последней строкой страницы и первой строкой следующей страницы. Значение n должно находиться в пределах 1...127 строк.

ESC O - Отмена режима пропуска перфорации
1Bh 4Fh
Эта команда отменяет режим пропуска перфорации, установленный командой "ESC N n".

LF(10) - Перевод строки
0Ah
Команда вызывает распечатку содержимого буфера данных принтера, после чего головка переводится в начало следующей строки (бумага продвигается вперед на одну строку).

ESC 0 - Выбор межстрочного интервала, равного 1/8 дюйма
1Bh 30h
Расстояние между текстовыми строками устанавливается равным 1/8 дюйма.

ESC 2 - Выбор межстрочного интервала, равного 1/6 дюйма
1Bh 32h
Расстояние между текстовыми строками устанавливается равным 1/6 дюйма. Это значение используется по умолчанию при включении питания принтера.

ESC 3 n - Выбор межстрочного интервала, равного n/180 дюйма
1Bh 33h n
Расстояние между текстовыми строками устанавливается равным n/180 дюйма. Значение n должно находиться в пределах 0...255.

ESC + n - Выбор межстрочного интервала, равного n/360 дюйма
1Bh 2Bh n
Расстояние между текстовыми строками устанавливается равным n/360 дюйма. Значение n должно находиться в пределах 0...255.

ESC A n - Выбор межстрочного интервала, равного n/60 дюйма
1Bh 41h n
Расстояние между текстовыми строками устанавливается равным n/60 дюйма. Значение n должно находиться в пределах 0...85.

ESC J n - Проброс бумаги на расстояние n/180 дюйма
1Bh 4Ah n
Бумага продвигается вперед на расстояние, равное n/180 дюймов. Команда выполняется немедленно и не вызывает перемещений печатающей головки.

VT(11) - Вертикальная табуляция
0Bh
Бумага продвигается до следующего символа табуляции в канале, выбранном командой "ESC /". Если не выбран никакой канал, по умолчанию используется нулевой. Если вертикальная табуляция не установлена, бумага продвигается вперед на одну строку.

ESC B n1 n2 ... 0 - Установка вертикальной табуляции
1Bh 42h n1 n2 ... 00h
Команда позволяет задать до 16 положений для вертикальной табуляции. Параметры n1, n2, ... задают позиции для табуляции. Они должны указываться в порядке возрастания. Последний параметр всегда должен быть равен 0 - это признак конца последовательности параметров. Табуляция устанавливается в нулевом канале.

ESC b c n1 n2 ... 0 - Установка вертикальной табуляции в канале
1Bh 62h c 1n n2 ... 00h
Команда аналогична команде "ESC B", за исключением того, что необходимо указывать параметр c - номер выбираемого канала для вертикальной табуляции. Значение параметра c должно находиться в пределах 0...7.

ESC / c - Выбор канала для вертикальной табуляции
1Bh 2Fh c
Команда выбирает канал c для работы с командами VT вертикальной табуляции. Значение параметра c должно находиться в пределах 0...7.

ESC l n - Установка левой границы
1Bh 6Ch n
Устанавливается левая граница листа. В левой части листа оставляется n пустых столбцов символов текущей ширины. Если команда выдается для пропорционального набора символов, в качестве ширины для установки левой границы берется значение 10 символов на дюйм (10 pitch).

ESC Q n - Установка правой границы
1Bh 51h n
Устанавливается правая граница листа. В правой части листа оставляется n пустых столбцов символов текущей ширины. Если команда выдается для пропорционального набора символов, в качестве ширины для установки правой границы берется значение 10 символов на дюйм (10 pitch).

BS(8) - Возврат на одну позицию
08h
Команда вызывает распечатку содержимого буфера печати, после чего печатающая головка возвращается на одну позицию назад (но не за левую границу, установленную командой "ESC l". Если этот код выводится на принтер сразу после печати строки в графическом режиме, печатающая головка возвращается к началу только что напечатанной графической строки.

ESC $ n1 n2 - Установка абсолютной позиции для печати
1Bh 24h n1 n2
Команда задает расстояние от левой границы листа до того места, откуда будет продолжена печать символов. Для вычисления расстояния используется следующая формула: n1 + (n2 * 256). Расстояние задается в единицах, эквивалентных 1/60 доли дюйма.

ESC \ n1 n2 - Установка относительной позиции для печати
1Bh 5Ch n1 n2
Команда задает расстояние от текущей позиции печатающей головки до того места, откуда будет продолжена печать символов. Для вычисления расстояния используется следующая формула: n1 + (n2 * 256). Расстояние задается в единицах, эквивалентных 1/120 доли дюйма для чернового режима и 1/180 доли дюйма для качественного и пропорционального.

HT(9) - Горизонтальная табуляция
09h
Печатающая головка продвигается до следующего символа горизонтальной табуляции. По умолчанию для одного символа табуляции используется интервал в 8 символов текущего размера.

ESC D n1 n2 ... 0 - Установка горизонтальной табуляции
1Bh 44h n1 n2 ... 00h
Команда позволяет задать до 32 положений для горизонтальной табу-

ляции. Параметры n1, n2, ... задают позиции для табуляции. Они должны указываться в порядке возрастания. Последний параметр всегда должен быть равен нулю - это признак конца последовательности параметров. Команда "ESC D 0" сбрасывает все позиции горизонтальной табуляции.

ESC x n - Выбор черновой или качественной печати

1Bh 78h n

Параметр n определяет режим печати следующим образом:

0 черновой режим печати
1 качественный (LQ) режим печати

ESC k n - Выбор стиля печати

1Bh 6Bh n

Команда действительна только для режима качественной печати. Параметр n задает стиль:

0 - Roman;
1 - Sans Serif;
2 - Courier;
3 - Prestige;
4 - Script;
5 - OCR-B;
6 - OCR-A.

ESC ! n - Выбор режима работы принтера

1Bh 21h n

Команда позволяет задать комбинацию различных режимов работы принтера. Можно по отдельности задавать размер символов (10 или 12 символов на дюйм), набор символов (пропорциональный, сжатый, выделенный и т. д.). Отдельные биты байта параметра n задают режим работы принтера следующим образом:

ESC P - Выбор размера символа в 10 pitch

1Bh 50h

Эта команда задает размер символа, равный 10 pitch, или 10 символов на дюйм. Такой размер устанавливается по умолчанию при инициализации принтера.

ESC M - Выбор размера символа в 12 pitch
1Bh 4Dh
Эта команда задает размер символа, равный 12 pitch, или 12 символов на дюйм.

ESC g - Выбор размера символа в 15 pitch
1Bh 67h
Эта команда задает размер символа, равный 15 pitch, или 15 символов на дюйм. Этот режим не совместим с режимом сжатой печати.

ESC p n - Включение/выключение пропорционального режима
1Bh 70h n
В пропорциональном режиме разные символы имеют различную ширину, что благоприятно сказывается на читаемости текста. Например, букава "i" уже, чем "W".
Параметр n может принимать следующие значения:

0 - выключение пропорционального режима;
1 - включение пропорционального режима.

SI(15) - Выбор режима сжатой печати
0Fh
В этом режиме символы имеют примерно на 60 процентов меньшую ширину, чем в нормальном режиме. Режим сжатой печати не совместим с пропорциональным режимом.

ESC SI - Выбор режима сжатой печати
1Bh 0Fh
Команда полностью аналогична предыдущей команде SI.

DC2(18) - Отмена режима сжатой печати
12h
Отменяется режим сжатой печати, установленный ранее командами ESC SI или SI.

SO(14) - Печать с двойной шириной
0Eh
В этом режиме ширина каждого распечатываемого символа увеличивается в два раза. Режим отменяется командой возврата каретки или командой DC4.

ESC SO - Печать с двойной шириной
1Bh 0Eh
Команда полностью аналогична предыдущей команде SO.

ESC W n - Включение/выключение режима печати с двойной высотой
1Bh 77h n
В режиме печати с двойной высотой высота каждого распечатываемого символа увеличивается в два раза.
Возможные значения параметра n:

0 - выключение режима печати с двойной высотой;
1 - включение режима печати с двойной высотой.

DC4(20) - Отмена режима печати с двойной шириной
14h
Команда отменяет действие команд ESC SO или SO, но не действует, если режим печати с двойной шириной задан командами ESC W или ESC !.

ESC W n - Включение/выключение режима печати с двойной шириной
1Bh 57h n
В режиме печати с двойной шириной ширина каждого распечатываемого символа увеличивается в два раза.
Возможные значения параметра n:

0 - выключение режима печати с двойной шириной;
1 - включение режима печати с двойной шириной.

ESC E - Установка режима печати с выделением
1Bh 45h
Распечатываемые символы выглядят "толще" за счет того, что каждая точка печатается дважды.

ESC F - Отмена режима печати с выделением
1Bh 46h
Команда отменяет действие команды ESC E.

ESC G - Установка режима двойной печати
1Bh 47h
В режиме двойной печати каждая строка печатается дважды, поэтому текст выглядит ярче. Скорость печати уменьшается в два раза.

ESC H - Отмена режима двойной печати
1Bh 48h
Команда отменяет действие команды ESC G.

ESC S 0 - Печать верхнего индекса
1Bh 53h 00h
Символы распечатываются выше обычного уровня, занимая верхние две трети сетки.

ESC S 1 - Печать нижнего индекса
1Bh 53h 01h
Символы распечатываются ниже обычного уровня, занимая нижние две трети сетки.

ESC T - Отмена печати верхнего или нижнего индекса
1Bh 54h
Команда отменяет действие любой из команд, задающих режим печати индекса, - ESC S 0 или ESC S 1.

ESC - n - Включение/выключение режима подчеркивания
1Bh 2Dh n
В зависимости от значения параметра n все символы (и пробелы тоже) печатаются с подчеркиванием или без подчеркивания:

0 - выключение режима подчеркивания;
1 - включение режима подчеркивания.

ESC q n - Выбор стиля распечатываемых символов
1Bh 71h n
В зависимости от значения параметра n все символы, кроме имеющих коды от B0h до DFh и символа с кодом F5h, распечатываются с использованием следующих стилей:

0 - обычный стиль;
1 - контурное (outline) начертание символов;
2 - использование тени (стиль shadow);
3 - комбинация контурного начертания и тени.

Обработка слов

ESC a n - Выравнивание для качественного (LQ) набора символов
1Bh 61h n
Параметр n может принимать следующие значения:

0 - выравнивание влево;
1 - выравнивание по центру;
2 - выравнивание вправо;
3 - полное выравнивание.

По умолчанию при инициализации принтера выбирается режим выравнивания влево. Полное выравнивание выполняется после заполнения буфера печати. При этом распечатываемый текст может содержать символы горизонтальной табуляции HT и возврата на одну позицию BS только тогда, когда задан режим выравнивания влево (n = 0). Если используется полное выравнивание, параграфы текста не должны содержать символы возврата каретки.

ESC SP n - Выбор расстояния между символами
1Bh 20h n
Команда позволяет увеличить расстояние между символами по сравнению с тем, которое было задано в сетке при разработке начертания символов. Параметр n, значение которого должно лежать в пределах 0...127, задает количество точек, добавляемых справа к каждому символу. Одна точка соответствует 1/120 дюйма в черновом режиме и 1/180 дюйма в качественном и пропорциональном режимах.

Таблицы символов

ESC t n - Выбор таблицы символов
1Bh 74h n
Данная команда выбирает одну из таблиц, описывающих начертание символов для символов с кодами от 128 до 255, т. е. для правой половины кодовой таблицы ASCII. Можно выбрать таблицу, содержащую символы курсива (только латинские буквы), расширенную графическую таблицу фирмы Epson или таблицу, определенную пользователем:

0 - таблица с символами курсива (Italics character table);
1 - расширенная графическая таблица Epson;
2 - таблица, определенная пользователем.

После загрузки в принтер русских (или других пользовательских) шрифтов необходимо задать режим n = 2.

ESC 4 - Использование курсива
1Bh 34h
После получения этой команды принтер печатает весь текст курсивом для любой таблицы символов, за исключением символов псевдографики расширенной таблицы Epson.

ESC 5 - Отмена использования курсива
1Bh 35h
Команда отменяет печать курсивом.

ESC R n - Выбор национального набора символов
1Bh 52h n
В зависимости от параметра n выбирается национальный набор символов:

0 - США;
1 - Франция;
2 - Германия;
3 - Англия;
4 - Дания, набор 1;
5 - Швеция;
6 - Италия;
7 - Испания, набор 1;
8 - Япония;
9 - Норвегия;
10 - Дания, набор 2;
11 - Испания, набор 2;
12 - Латинская Америка.

ESC & - Определение символов
1Bh 26h 00h n1 n2 d0 d1 d2 data
Полный формат команды для переопределения символов в принтере Epson LQ-2550:
ESC "&" "0" n1 n2 d0 d1 d2 data
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 зада-

ет количество свободных столбцов слева от символа, 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.

ESC : 0 n 0 - Копирование символов из ПЗУ в ОЗУ
1Bh 3Ah 00h n 00h
Команда выполняет копирование заданного параметром n набора символов из постоянного запоминающего устройства принтера в его оперативную память. После этого можно переопределить начертание части символов командой ESC &. Возможные значения параметра n:

0 - Roman;
1 - Sans Serif;
2 - Courier;
3 - Prestige;
4 - Script;
5 - OCR-B;
6 - OCR-A.

ESC % n - Выбор набора символов, заданного пользователем
1Bh 25h n
Команда позволяет переключать используемый набор символов. В зависимости от значения параметра n будет использоваться либо стандартный набор символов принтера, либо набор символов, определенный пользователем при помощи команды ESC &.
Возможные значения параметра n:

0 - используется стандартный набор символов;
1 - используется набор символов, определенный пользователем.

ESC 6 - Разрешение печати символов с кодами 128...159
1Bh 36h
После приема этой команды при использовании расширенной графической таблицы Epson символы с кодами от 128 до 159 будут интерпретироваться принтером как символы, а не как управляющие коды.

ESC 7 - Запрещение печати символов с кодами 128...159
1Bh 37h
После приема этой команды при использовании расширенной графической таблицы Epson символы с кодами от 128 до 159 будут интерпретироваться принтером как управляющие коды. Этот режим устанавливается по умолчанию при инициализации принтера.

Графические команды

ESC K n1 n2 - Выбор графического режима с одинарной плотностью
1Bh 4Bh n1 n2
Команда устанавливает графический 8-битовый режим одинарной плотности. Общее количество столбцов в графической строке при этом составит n1 + (n2 * 256).

ESC L n1 n2 - Выбор графического режима с двойной плотностью
1Bh 4Ch n1 n2
Команда устанавливает графический 8-битовый режим двойной плотности, печать будет выполняться с низкой скоростью. Общее количество столбцов в графической строке при этом n1 + (n2 * 256).

ESC Y n1 n2 - Выбор скоростного графического режима с двойной плотностью
1Bh 59h n1 n2
Команда устанавливает графический 8-битовый режим двойной плотности, печать будет выполняться с высокой скоростью. Общее количество столбцов в графической строке при этом n1 + (n2 * 256).

ESC Z n1 n2 - Выбор графического режима с учетверенной плотностью
1Bh 5Ah n1 n2
Команда устанавливает графический 8-битовый режим учетверенной плотности. Общее количество столбцов в графической строке при этом составит n1 + (n2 * 256).

ESC * - Печать в графическом режиме
1Bh 2Ah m n1 n2
Полный формат команды графической печати:
ESC "*" m n1 n2 data
В этой команде параметр m задает режим печати:

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

ESC ? s n - Переназначение графических режимов

1Bh 3Fh s n

Команда позволяет заменить один графический режим на другой. Параметр s - это символ (K, L, Y, Z), который назначается режиму, заданному параметром n (0...6).

Управление цветом

ESC r n - Определение цвета печати

1Bh 72h n

В зависимости от значения параметра n печать будет выполнена одним из следующих цветов:

0 - черный;
1 - малиновый;
2 - бирюзовый;
3 - фиолетовый;
4 - желтый;
5 - красный;
6 - зеленый.

Приложение 8

КОМАНДЫ ПРИНТЕРОВ Epson FX-850/1050

Управление принтером

ESC @ - Инициализация принтера
1Bh 40h
Epson
Выполняется сброс принтера в исходное состояние, если происходила печать строки, то при инициализации эта печать отменяется.

DC1(17) - Выбор принтера для работы
11h
Epson, IBM
Возвращает принтер в выбранное (активное) состояние, если раньше он был переведен в неактивное состояние командой DC3. Эта команда не выполняет никаких действий, если принтер находится в состоянии OFF LINE

DC3(19) - Перевод принтера в неактивное состояние
13h
Epson
Переводит принтер в неактивное состояние до получения принтером команды DC1 выбора принтера для работы. Если принтер переведен в неактивное состояние, его нельзя перевести в состояние ON LINE при помощи соответствующей клавиши на лицевой панели.

DEL(127) - Удаление символа
7Fh
Epson
Удаляет последний переданный в принтер символ. Эта команда не действует на управляющие коды.

ESC s n - Включение/выключение половинной скорости печати
1Bh 73h n
Epson
Значения параметра n:

0 - режим половинной скорости печати выключен;
1 - режим половинной скорости печати включен.

ESC < - Печать в одном направлении
Epson
1Bh 3Ch
Переключает принтер в режим печати в одном направлении, т. е. принтер печатает только тогда, когда печатающая головка движется в

одну сторону. Команда задает режим только для одной строки, символ возврата каретки отменяет печать только в одном направлении.

ESC U n - Включение/выключение режима печати в одномнаправлении
1Bh 55h n
Epson, IBM
В зависимости от параметра n режим печати только в одном направлении может включаться либо отключаться:

n = 1 режим включен;
n = 0 режим выключен.

ESC EM n - Управление податчиком бумаги
1Bh 19h n
Epson
Эта команда распознается принтером только в том случае, если принтер оборудован податчиком бумаги (Cut Sheet Feeder).
Возможные значения параметра n:

0 отключение податчика бумаги;
4 включение податчика бумаги.

ESC 8 - Отмена проверки конца бумаги
1Bh 38h
Epson
После приема этой команды принтер не останавливает печать даже в том случае, если кончилась бумага.

ESC 9 - Включение проверки конца бумаги
1Bh 39h
Epson
Команда отменяет действие предыдущей команды.

BEL(7) - Выдача звукового сигнала
07h
Epson, IBM
Если выдать эту команду на принтер, громкоговоритель принтера издаст звуковой сигнал. Эта команда может быть использована для предупреждения оператора, например, о том, что кончилась бумага.

Управление старшим битом данных

Старший бит данных MSB (Most Significant Bit) - это бит 7 байтов, посылаемых программой в принтер. Обычно Вам не надо изменять значение старшего бита данных, более того, команды управления старшим битом не действуют в графическом режиме или при переопределении символов.

ESC = - Установка MSB в 0
1Bh 3Dh
Epson
Для всех выводимых на принтер данных старший бит принудительно

устанавливается в 0. Этот режим используется при работе принтера с такими компьютерами, которые всегда посылают на принтер данные со старшим битом, установленным в 1.

ESC > - Установка MSB в 1
1Bh 3Eh
Epson
Для всех выводимых на принтер данных старший бит принудительно устанавливается в 1.

ESC # - Отмена управления MSB
1Bh 23h
Epson
Отменяется действие введенных ранее команд управления MSB "ESC =" или "ESC >"

Управление печатающей головкой и перемещением бумаги

CR(13) - Возврат каретки
0Dh
Epson, IBM
Команда вызывает распечатку содержимого буфера данных принтера, после чего головка переводится в начало текущей строки.

CAN(24) - Отмена печати строки
18h
Epson, IBM
Текст из распечатываемой строки удаляется. Команда не влияет на управляющие коды, находящиеся в этой строке.

FF(12) - Подача бумаги на один лист
0Ch
Epson, IBM
Команда вызывает распечатку содержимого буфера принтера, вслед за этим происходит подача бумаги вперед на один лист (в соответствии с установленной длиной листа).

ESC C n - Установить длину листа бумаги в строках
1Bh 43h n
Epson, IBM
Команда устанавливает длину листа бумаги равной n строкам. Используется установленное ранее расстояние между строками. Допустимые значения параметра n лежат в пределах 1...127 строк.

ESC C 0 n - Установить длину листа бумаги в дюймах
1Bh 43h 00h n
Epson, IBM
Команда устанавливает длину листа бумаги равной n дюймам. Значение n находиться в пределах 1...22 дюймов.

ESC N n - Установить режим пропуска перфорации
1Bh 4Eh n
Epson, IBM
В этой команде параметр n - это количество строк, пропускаемых принтером между последней строкой страницы и первой строкой следующей страницы. Значение n должно находиться в пределах 1...127 строк.

ESC O - Отмена режима пропуска перфорации
1Bh 4Fh
Epson, IBM
Эта команда отменяет режим пропуска перфорации, установленный командой ESC N n.

ESC 4 - Установка верхней границы листа
1Bh 34h
IBM
Текущая строка становится верхней строкой листа.

LF(10) - Перевод строки
0Ah
Epson, IBM
Команда вызывает распечатку содержимого буфера данных принтера, после чего головка переводится в начало следующей строки (бумага продвигается вперед на одну строку).

ESC 0 - Выбор межстрочного интервала, равного 1/8 дюйма
1Bh 30h
Epson, IBM
Расстояние между текстовыми строками устанавливается равным 1/8 дюйма.

ESC 1 - Выбор межстрочного интервала, равного 7/72 дюйма
1Bh 31h
Epson, IBM
Расстояние между текстовыми строками устанавливается равным 7/72 дюйма.

ESC 2 - Выбор межстрочного интервала, равного 1/6 дюйма
1Bh 32h
Epson, IBM
Расстояние между текстовыми строками устанавливается равным 1/6 дюйма. Это значение используется по умолчанию при включении питания принтера.

ESC 2 - Выбор межстрочного интервала командой ESC A n
1Bh 32h
IBM
После приема принтером этой команды межстрочный интервал устанавливается таким, каким он был задан в команде ESC A n.

ESC 3 n - Выбор межстрочного интервала, равного n/216 дюйма
1Bh 33h n
Epson, IBM
Расстояние между текстовыми строками устанавливается равным n/216 дюйма. Значение n должно находиться в пределах 0...255.

ESC A n - Выбор межстрочного интервала, равного n/72 дюйма
1Bh 41h n
Epson
Расстояние между текстовыми строками устанавливается равным n/72 дюйма. Значение n должно находиться в пределах 0...85.

ESC A n - Выбор межстрочного интервала, равного n/72 дюйма
1Bh 41h n
IBM
Расстояние между текстовыми строками устанавливается равным n/72 дюйма. Значение n должно находиться в пределах 0...85. Для того чтобы эта команда включилась в работу, принтер должен принять команду ESC 2.

ESC J n - Проброс бумаги на расстояние n/216 дюйма
1Bh 4Ah n
Epson, IBM
Бумага продвигается вперед на расстояние, равное n/216 дюйма. Команда выполняется немедленно и не вызывает перемещений печатающей головки. Диапазон допустимых значений n - 0...255.

ESC 5 - Переключение режима автоматического перевода строки
1Bh 35h n
IBM
Если включен режим автоматического перевода строки, то принтер переводит строку после приема символа возврата каретки CR без дополнительного символа перевода каретки LF. Возможные значения параметра n:

0 - выключить автоматический перевод строки;
1 - включить автоматический перевод строки.

VT(11) - Вертикальная табуляция
0Bh
Epson, IBM
Бумага продвигается до следующей позиции вертикальной табуляции. Если вертикальная табуляция не установлена, бумага продвигается вперед на одну строку.

ESC B n1 n2 ... 0 - Установка вертикальной табуляции
1Bh 42h n1 n2 ... 00h
Epson, IBM
Команда позволяет задать до 16 положений для вертикальной табуляции. Параметры n1, n2, ... задают позиции для табуляции. Они должны указываться в порядке возрастания. Последний параметр всегда

должен быть равен нулю - это признак конца последовательности параметров.

ESC b c n1 n2 ... 0 - Установка вертикальной табуляции в канале
1Bh 62h c 1n n2 ... 00h
Epson
Команда аналогична команде ESC B, за исключением того, что необходимо указывать параметр c - номер выбираемого канала для вертикальной табуляции. Значение параметра c должно находиться в пределах 0...7.

ESC / c - Выбор канала для вертикальной табуляции
1Bh 2Fh c
Epson
Команда выбирает канал c для работы с командами VT вертикальной табуляции. Значение параметра c должно находиться в пределах 0...7.

ESC l n - Установка левой границы
1Bh 6Ch n
Epson
Устанавливается левая граница листа. В левой части листа оставляется n пустых столбцов символов текущей ширины. Если командв выдается для пропорционального набора символов, в качестве ширины для установки левой границы берется значение 10 символов на дюйм (10 pitch).

ESC Q n - Установка правой границы
1Bh 51h n
Epson
Устанавливается правая граница листа. В правой части листа оставляется n пустых столбцов символов текущей ширины. Если команда выдается для пропорционального набора символов, в качестве ширины для установки правой границы берется значение 10 символов на дюйм (10 pitch).

ESC X - Установка левой и правой границы листа
1Bh 58h n1 n2
IBM
Левая граница устанавливается в n1, правая в n2. Используется текущая ширина символов.

BS(8) - Возврат на одну позицию
08h
Epson, IBM
Команда вызывает распечатку содержимого буфера печати, после чего печатающая головка возвращается на одну позицию назад (но не за левую границу, установленную командой "ESC l". После команд "ESC a 2" и "ESC a 3" эта команда игнорируется. Если команда BS выводится на принтер сразу после печати строки в графическом режи-

ме, печатающая головка возвращается к началу только что напечатанной графической строки.

ESC $ n1 n2 - Установка абсолютной позиции для печати
1Bh 24h n1 n2
Epson, IBM
Команда задает расстояние от левой границы листа до того места, откуда будет продолжена печать символов. Для вычисления расстояния используется следующая формула: n1 + (n2 * 256). Расстояние задается в единицах, эквивалентных 1/60 доли дюйма.

ESC \ n1 n2 - Установка относительной позиции для печати
1Bh 5Ch n1 n2
Epson
Команда задает расстояние от текущей позиции печатающей головки до того места, откуда будет продолжена печать символов. Для вычисления расстояния вначале надо вычислить требуемое смещение в точках. Если надо определить смещение влево, полученное значение следует вычесть из числа 65536. Общее количество точек вычисляется по формуле n1 + (n2 * 256).

HT(9) - Горизонтальная табуляция
09h
Epson, IBM
Печатающая головка продвигается до следующего символа горизонтальной табуляции.

ESC D n1 n2 ... 0 - Установка горизонтальной табуляции
1Bh 44h n1 n2 ... 00h
Epson, IBM
Команда позволяет задать до 32 положений для горизонтальной табуляции. Параметры n1, n2, ... задают позиции для табуляции. Они должны указываться в порядке возрастания. Последний параметр всегда должен быть равен нулю - это признак конца последовательности параметров. Команда "ESC D 0" сбрасывает все позиции горизонтальной табуляции. После включения питания (или приема команды инициализации "ESC @") каждому символу табуляции при печати будет соответствовать восемь пробелов.

ESC R - Отмена табуляции
1Bh 52h
IBM
Отменяется одновременно горизонтальная и вертикальная табуляция.

ESC x n - Выбор черновой или качественной печати
1Bh 78h n
Epson
Параметр n определяет режим печати следующим образом:

0 - черновой режим печати;
1 - качественный (NLQ) режим печати.

В качественном режиме печати используются наборы символов Roman или Sans Serif.

ESC k n - Выбор стиля печати

1Bh 6Bh n

Epson

Команда действительна только для режима качественной печати. Параметр n задает стиль:

0 - Roman;
1 - Sans Serif.

ESC ! n - Выбор режима работы принтера

1Bh 21h n

Epson

Команда позволяет задать комбинацию различных режимов работы принтера. Можно по отдельности задавать размер символов (10 или 12 символов на дюйм), набор символов (пропорциональный, сжатый, выделенный и т. д.). Отдельные биты байта параметра n задают режим работы принтера следующим образом:

ESC I n - Выбор шрифта

1Bh 49h n

IBM

В зависимости от параметра n выбирается один из шрифтов:

0 - обычное качество, 10 символов на дюйм;
1 - обычное качество, 12 символов на дюйм;
2 - качественный шрифт Sans Serif;
3 - качественный шрифт Roman;
4 - шрифт обычного качества, определенный пользователем, 10 символов на дюйм;
5 - шрифт обычного качества, определенный пользователем, 12 символов на дюйм;
6 - качественный шрифт, определенный пользователем, 10 символов на дюйм;
7 - качественный шрифт, определенный пользователем, 12 символов на дюйм.

ESC P - Выбор размера символа в 10 pitch
1Bh 50h
Epson
Эта команда задает размер символа, равный 10 pitch, или 10 символов на дюйм. Такой размер устанавливается по умолчанию при инициализации принтера.

ESC M - Выбор размера символа в 12 pitch
1Bh 4Dh
Epson
Эта команда задает размер символа, равный 12 pitch, или 12 символов на дюйм.

DC2(18) - Выбор размера символа в 10 pitch
12h
IBM
Эта команда задает размер символа, равный 10 pitch, или 10 символов на дюйм. Команда отменяет ранее установленный режим сжатой печати.

ESC : - Выбор размера символа в 12 pitch
1Bh 3Ah
IBM
Эта команда задает размер символа, равный 10 pitch, или 12 символов на дюйм.

ESC p n - Включение/выключение пропорционального режима
1Bh 70h n
Epson
В пропорциональном режиме разные символы имеют различную ширину, что благоприятно сказывается на читаемости текста. Например, букава "i" уже, чем "W".
Параметр n может принимать следующие значения:

0 - выключение пропорционального режима;
1 - включение пропорционального режима.

ESC P n - Включение/выключение пропорционального режима
1Bh 50h n
IBM
Параметр n может принимать следующие значения:

0 - выключение пропорционального режима;
1 - включение пропорционального режима.

SI(15) - Выбор режима сжатой печати
0Fh
Epson, IBM
В этом режиме символы имеют примерно на 60 процентов меньшую ширину, чем в нормальном режиме. Режим сжатой печати не совместим с пропорциональным режимом.

ESC SI - Выбор режима сжатой печати
1Bh 0Fh
Epson, IBM
Команда полностью аналогична предыдущей команде SI.

DC2(18) - Отмена режима сжатой печати
12h
Epson
Отменяется режим сжатой печати, установленный ранее командами ESC SI или SI.

SO(14) - Печать с двойной шириной
0Eh
Epson, IBM
В этом режиме ширина каждого распечатываемого символа увеличивается в два раза. Режим печати с двойной шириной отменяется командой возврата каретки или командой DC4.

ESC SO - Печать с двойной шириной
1Bh 0Eh
Epson, IBM
Команда полностью аналогична предыдущей команде SO.

ESC W n - Включение/выключение режима печати с двойной высотой
1Bh 77h n
Epson, IBM
В режиме печати с двойной высотой высота каждого распечатываемого символа увеличивается в два раза.
Возможные значения параметра n:

0 - выключение режима печати с двойной высотой;
1 - включение режима печати с двойной высотой.

DC4(20) - Отмена режима печати с двойной шириной
14h
Epson, IBM
Команда отменяет действие команд ESC SO или SO, но не действует, если режим печати с двойной шириной задан командами ESC Wили ESC !.

ESC W n - Включение/выключение режима печати с двойной шириной
1Bh 57h n
Epson
В режиме печати с двойной шириной ширина каждого распечатываемого символа увеличивается в два раза.
Возможные значения параметра n:

0 - выключение режима печати с двойной шириной;
1 - включение режима печати с двойной шириной.

ESC [@ - Печать с двойной высотой и двойной шириной
1Bh 5Bh 40h n1 n2 m1 m2 m3 m4

Команда позволяет управлять высотой и шириной распечатываемых символов. Параметры n2, m1, m2 должны быть равны 0, параметр n1 должен быть равен 4. Параметр m3 влияет на высоту символов и количество пробрасываемых по команде LF строк:

1 - стандартная высота, текущий режим перевода строки;
2 - двойная высота, обычный режим перевода строки;
16 - текущая высота, одна строка на один LF;
17 - стандартная высота, одна строка на один LF;
18 - двойная высота, одна строка на один LF;
32 - текущая высота, две строки на один LF;
33 - стандартная высота, две строки на один LF;
34 - двойная высота, две строки на один LF;

Параметр m4 влияет на ширину символов:

1 - стандартная ширина;
2 - двойная ширина.

ESC E - Установка режима печати с выделением
1Bh 45h
Epson, IBM
Распечатываемые символы выглядят "толще" за счет того, что каждая точка печатается дважды.

ESC F - Отмена режима печати с выделением
1Bh 46h
Epson, IBM
Команда отменяет действие команды ESC E.

ESC G - Установка режима двойной печати
1Bh 47h
Epson, IBM
В режиме двойной печати каждая строка печатается дважды, поэтому текст выглядит ярче. Скорость печати уменьшается в два раза.

ESC H - Отмена режима двойной печати
1Bh 48h
Epson, IBM
Команда отменяет действие команды ESC G.

ESC S 0 - Печать верхнего индекса
1Bh 53h 00h
Epson, IBM
Символы распечатываются выше обычного уровня, занимая верхние две трети сетки.

ESC S 1 - Печать нижнего индекса
1Bh 53h 01h
Epson, IBM
Символы распечатываются ниже обычного уровня, занимая нижние две трети сетки.

ESC T - Отмена печати верхнего или нижнего индекса
1Bh 54h
Epson, IBM
Команда отменяет действие любой из команд, задающих режим печати индекса, - ESC S 0 или ESC S 1.

ESC - n - Включение/выключение режима подчеркивания
1Bh 2Dh n
Epson, IBM
В зависимости от значения параметра n все символы (и пробелы тоже) печатаются с подчеркиванием или без подчеркивания:

0 - выключение режима подчеркивания;
1 - включение режима подчеркивания.

ESC _ n - Включение/выключение режима перечеркивания
1Bh 2Dh n
IBM
В зависимости от значения параметра n все символы печатаются перечеркнутыми или нет:

0 - выключение режима перечеркивания;
1 - включение режима перечеркивания.

Обработка слов

ESC a n - Выравнивание для качественного (LQ) набора символов
1Bh 61h n
Epson
Параметр n может принимать следующие значения:

0 - выравнивание влево;
1 - выравнивание по центру;
2 - выравнивание вправо;
3 - полное выравнивание.

По умолчанию при инициализации принтера выбирается режим выравнивания влево. Полное выравнивание выполняется после заполнения буфера печати. При этом распечатываемый текст может содержать символы горизонтальной табуляции HT и возврата на одну позицию BS только тогда, когда задан режим выравнивания влево (n = 0). Если используется полное выравнивание, параграфы текста не должны содержать символы возврата каретки.

ESC SP n - Выбор расстояния между символами
1Bh 20h n
Epson
Команда позволяет увеличить расстояние между символами по сравнению с тем, которое было задано в сетке при разработке начертания символов. Параметр n, значение которого должно лежать в пределах 0...127, задает количество точек, добавляемых справа к каждому символу. Одна точка соответствует 1/120 дюйма.

Таблицы символов

ESC t n - Выбор таблицы символов
1Bh 74h n
Epson
Данная команда выбирает одну из таблиц, описывающих начертание символов для символов с кодами от 128 до 255, т. е. для правой половины кодовой таблицы ASCII. Можно выбрать таблицу, содержащую символы курсива (только латинские буквы) или расширенную графическую таблицу фирмы Epson:

0 - таблица с символами курсива (Italics character table);
1 - расширенная графическая таблица Epson.

ESC 4 - Использование курсива
1Bh 34h
Epson
После получения этой команды принтер печатает весь текст курсивом для любой таблицы символов, за исключением символов псевдографики расширенной таблицы Epson.

ESC 5 - Отмена использования курсива
1Bh 35h
Epson
Команда отменяет печать курсивом.

ESC R n - Выбор национального набора символов
1Bh 52h n
Epson
В зависимости от параметра n выбирается национальный набор символов:

0 - США;
1 - Франция;
2 - Германия;
3 - Англия;
4 - Дания, набор 1;
5 - Швеция;
6 - Италия;
7 - Испания, набор 1;
8 - Япония;
9 - Норвегия;
10 - Дания, набор 2;
11 - Испания, набор 2;
12 - Латинская Америка.

ESC \ - Печать символов с кодами, меньшими 32
1Bh 5Ch n1 n2
IBM
Разрешается печать следующих за командой (n2 * 256) + n1 символов, имеющих коды, меньшие чем 32.

ESC ^ - Печать одного символа с кодом, меньшим 32
1Bh 5Eh n
IBM
Разрешается печать следующих за командой (n2 * 256) + n1 символов, имеющих коды, меньшие чем 32.

ESC & - Определение символов
1Bh 26h 00h d1 d2 ... dn
Epson
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 седьмом столбце. Тогда десятичное значение параметра a1 вычисляется следующим 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 a1 = 8.

Параметры d1...dn - образцы столбцов точек для определяемого символа. Их должно быть всегда 11, даже если символ содержит пустые столбцы. Для пустых столбцов в качестве образца надо задать нуль. 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.

ESC = - Определение символов
1Bh 26h 00h n1 n2 ... nk
IBM
Команда предназначена для переопределения символов. Для определения ее параметров можно воспользоваться следующей методикой:

- пусть C - общее количество переопределяемых символов;

- вычисляем B по формуле B=(C*13)+2;
- n1 = B MOD 256;
- n2 = INT(B/256);
- n3 = 20 (всегда);
- n4 = код первого переопределяемого символа;
- n5 = 0, если используются верхних восемь иголок печатающей головки, n5 = 128, если используются нижних восемь иголок;
- n6 = 0 (всегда);
- n7...nk = байты данных, определяющие начертание символа, их должно быть 11.

Для включения переопределенных символов в работу необходимо выдать принтеру команду ESC I.

ESC : 0 n 0 - Копирование символов из ПЗУ в ОЗУ
1Bh 3Ah 00h n 00h
Epson
Команда выполняет копирование заданного параметром n набора символов из постоянного запоминающего устройства принтера в его оперативную память. После этого можно переопределить начертание части символов командой ESC &. Возможные значения параметра n:

0 - Roman;
1 - Sans Serif.

ESC % n - Выбор набора символов, заданного пользователем
1Bh 25h n
Epson
Команда позволяет переключать используемый набор символов. В зависимости от значения параметра n будет использоваться либо стандартный набор символов принтера, либо набор символов, определенный пользователем при помощи команды ESC &.
Возможные значения параметра n:

0 - используется стандартный набор символов;
1 - используется набор символов, определенный пользователем.

ESC 6 - Разрешение печати символов с кодами 128...159
1Bh 36h
Epson, IBM
После приема этой команды при использовании расширенной графической таблицы Epson символы с кодами от 128 до 159 будут интерпретироваться принтером как символы, а не как управляющие коды.

ESC 7 - Запрещение печати символов с кодами 128...159
1Bh 37h
Epson, IBM
После приема этой команды при использовании расширенной графической таблицы Epson символы с кодами от 128 до 159 будут интерпретироваться принтером как управляющие коды. Этот режим устанавливается по умолчанию при инициализации принтера.

Графические команды

ESC K n1 n2 - Выбор графического режима с одинарной плотностью
1Bh 4Bh n1 n2
Epson, IBM
Команда устанавливает графический 8-битовый режим одинарной плотности. Общее количество столбцов в графической строке при этом составит n1 + (n2 * 256).

ESC L n1 n2 - Выбор графического режима с двойной плотностью
1Bh 4Ch n1 n2
Epson, IBM
Команда устанавливает графический 8-битовый режим двойной плотности, печать будет выполняться с низкой скоростью. Общее количество столбцов в графической строке при этом n1 + (n2 * 256).

ESC Y n1 n2 - Выбор скоростного графического режима с двойной плотностью
1Bh 59h n1 n2
Epson, IBM
Команда устанавливает графический 8-битовый режим двойной плотности, печать будет выполняться с высокой скоростью. Общее количество столбцов в графической строке при этом n1 + (n2 * 256).

ESC Z n1 n2 - Выбор графического режима с учетверенной плотностью
1Bh 5Ah n1 n2
Epson, IBM
Команда устанавливает графический 8-битовый режим учетверенной плотности. Общее количество столбцов в графической строке при этом составит n1 + (n2 * 256).

ESC * - Печать в графическом режиме
1Bh 2Ah m n1 n2
Epson
Полный формат команды графической печати:
ESC "*" m n1 n2 data
В этой команде параметр m задает режим печати:

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
5	Режим плоттера, плотность 72 точек на дюйм, масштаб 1:1, 8-битовая графика
6	Режим CRT II, плотность 90 точек на дюйм, 8-битовая графика

7	Режим плоттера, плотность 144 точек на дюйм, масштаб 1:1, 8-битовая графика

ESC ? s n - Переназначение графических режимов

1Bh 3Fh s n

Epson

Команда позволяет заменить один графический режим на другой. Параметр s - это символ (K, L, Y, Z), который назначается графическому режиму, заданному параметром n (0...6).

ESC ^ - Выбор 9-битового графического режима

1Bh 5Eh m n1 n2

Epson

Команда позволяет использовать для графической печати все 9 иголок печатающей головки. Параметр m определяет плотность печати:

0 - одинарная плотность;
1 - двойная плотность.

Параметры n1 и n2 определяют общую длину графической строки (т. е. количество распечатываемых столбцов). Эта длина вычисляется по формуле n1+(n2*256). В 9-битовом графическом режиме на каждый столбец требуются два байта графических данных.

Приложение 9

АЛЬТЕРНАТИВНАЯ ТАБЛИЦА КОДИРОВКИ

В настоящее время существует несколько вариантов кодировки русских букв (кириллицы) для операционной системы MS-DOS - основная, альтернативная, минская и т. д. Они отличаются в основном расположением русских букв и символов псевдографики. Однако наибольшее распространение получила альтернативная таблица кодировки, особенно после того как в 1989 году эта таблица была принята IBM в качестве стандартной для Советского Союза. Локализованная версия MS-DOS 4.01 содержит соответствующую кодовую страницу:

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00		☺	☻	♥	♦	♣	♠	•	◘	○	◙	♂	♀	♪	♫	☼
10	►	◄	↕	‼	¶	§	▬	↨	↑	↓	→	←	∟	↔	▲	▼
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	⌂
80	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
90	Р	С	Т	У	Ф	Х	Ц	Ч	С	Щ	Ъ	Ы	Ь	Э	Ю	Я
a0	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
b0	▒	▓	▓	|	┤	┤	╣	╗	╗	╣	▌	▐	▓	▓	▌	▐
c0	└	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	═	╬	╧
d0	▌	▀	▀	▀	└	┌	▀	▀	┼	┘	┌	▀	▀	▌	▌	▀
e0	р	с	т	у	ф	х	ц	ч	▀	ц	ъ	ы	ь	э	▀	я
f0	Ё	ё	Є	є	Ї	ї	Ў	ў	°	·	·	√	№	¤	▀	

Приложение 11

СОДЕРЖИМОЕ ФАЙЛА sysp.inc

```
;
; Это макроопределение печатает символы на экране
;
@@out_ch     MACRO c1,c2,c3,c4,c5,c6,c7,c8,c9,c10
          mov   ah,02h
          IRP   chr,<c1,c2,c3,c4,c5,c6,c7,c8,c9,c10>
          IFB   <chr>
          EXITM
          ENDIF
          mov   dl,chr
          int   21h
          ENDM
          ENDM
@@out_str MACRO
          mov   ah,9
          int   21h
          ENDM
; Макро для выдачи звукового сигнала
BEEP  MACRO
      mov bx,0
      mov ax, 0E07h
      int 10h
          ENDM
State87 struc
     cr dw ?
     sr dw ?
     tg dw ?
     cmdlo dw ?
     cmdhi dw ?
     oprlo dw ?
     oprhi dw ?
     st0 dt ?
     st1 dt ?
     st2 dt ?
     st3 dt ?
     st4 dt ?
     st5 dt ?
     st6 dt ?
     st7 dt ?
State87 ends
```

```
/* Префикс программного сегмента PSP */
typedef struct _PSP_ {
     unsigned char int20h[2];
     unsigned mem_top;
     unsigned char reserv1;
     unsigned char call_dsp[5];
     void far *term_adr;
     void far *cbrk_adr;
     void far *crit_err;
     unsigned parn_psp;
     unsigned char file_tab[20];
     unsigned env_seg;
     void far *ss_sp;
     unsigned max_open;
     void far *file_tba;
     unsigned char reserv2[24];
     unsigned char disp[3];
     unsigned char reserv3[9];
     unsigned char fcb1[16];
     unsigned char fcb2[20];
     unsigned char p_size;
     unsigned char parm[127];
} PSP;
/* Блок управления устройством DOS */
typedef struct _DDCB_ {
     unsigned char drv_num;
     unsigned char drv_numd;
     unsigned sec_size;
     unsigned char clu_size;
     unsigned char clu_base;
     unsigned boot_siz;
     unsigned char fat_num;
     unsigned max_dir;
     unsigned data_sec;
     unsigned hi_clust;
     unsigned char fat_size;
     char reserv1;
     unsigned root_sec;
     void far *drv_addr;
     unsigned char media;
     unsigned char acc_flag;
     struct _DDCB_ far *next;
     unsigned reserv2;
     unsigned built;
} DDCB;
```

```
/* Управляющий блок DOS для файлов */
typedef struct _DFCB_ {
     unsigned handl_num;
     unsigned char access_mode;
     unsigned reserv1;
     unsigned dev_info;
     void far *driver;
     unsigned first_clu;
     unsigned time;
     unsigned date;
     unsigned long fl_size;
     unsigned long offset;
     unsigned reserv2;
     unsigned reserv7;
     unsigned reserv3;
     char reserv4;
     char filename[11];
     char reserv5[6];
     unsigned ownr_psp;
     unsigned reserv6;
     unsigned last_clu;
     char reserv8[4];
} DFCB;
/* Таблица файлов DOS */
typedef struct _DFT_ {
     struct _DFT_ far *next;
     unsigned file_count;
     DFCB dfcb;
} DFT;
/* Управляющий блок дискового буфера BCB */
typedef struct _BCB_ {
     struct _BCB_ far *next;
     unsigned char drive;
     unsigned char flag;
     unsigned sect_num;
     unsigned reserv1;
     DDCB far *ddcb;
     unsigned reserv2;
} BCB;
/* Информация о диске */
typedef struct _DINFO_ {
     char path[64];
     unsigned reserv1;
     unsigned reserv2;
     unsigned char reserv3;
     DDCB far *ddcb;
     unsigned cdir_clu;
```

```
     unsigned reserv4;
     unsigned reserv5;
     unsigned reserv6;
     unsigned char reserv7[7];
} DINFO;

/* Заголовок EXE-программы */

typedef struct _EXE_HDR_ {
     unsigned signature;
     unsigned part_pag;
     unsigned file_size;
     unsigned rel_item;
     unsigned hdr_size;
     unsigned min_mem;
     unsigned max_mem;
     unsigned ss_reg;
     unsigned sp_reg;
     unsigned chk_summ;
     unsigned ip_reg;
     unsigned cs_reg;
     unsigned relt_off;
     unsigned overlay;
} EXE_HDR;

/* Таблица расположения сегментов EXE-программы */

typedef struct _RELOC_TAB_ {
     unsigned offset;
     unsigned segment;
} RELOC_TAB;

/* Конфигурация дисковой подсистемы */

typedef struct _DISK_CONFIG_ {
     int  n_floppy;
     int  n_hard;
     int  t_floppy1;
     int  t_floppy2;
     int  t_hard1;
     int  t_hard2;
} DISK_CONFIG;

/* Таблица параметров дискеты */

typedef struct _DPT_ {
     unsigned char srt_hut;
     unsigned char dma_hlt;
     unsigned char motor_w;
     unsigned char sec_size;
     unsigned char eot;
     unsigned char gap_rw;
     unsigned char dtl;
     unsigned char gap_f;
```

```
     unsigned char fill_char;
     unsigned char hst;
     unsigned char mot_start;
} DPT;

/* Таблица параметров диска */

typedef struct _HDPT_ {
     unsigned max_cyl;
     unsigned char max_head;
     unsigned srwcc;
     unsigned swpc;
     unsigned char max_ecc;
     unsigned char dstopt;
     unsigned char st_del;
     unsigned char fm_del;
     unsigned char chk_del;
     unsigned char reserve[4];
} HDPT;

/* Элемент таблицы разделов */

typedef struct _PART_ENTRY_ {
     unsigned char flag;
     unsigned char beg_head;
     unsigned beg_sec_cyl;
     unsigned char sys;
     unsigned char end_head;
     unsigned end_sec_cyl;
     unsigned long rel_sec;
     unsigned long size;
} PART_ENTRY;

/* Главная загрузочная запись */

typedef struct _MBOOT_ {
     char boot_prg[0x1be];
     PART_ENTRY part_table[4];
     unsigned char signature[2];
} MBOOT;

/* Расширенный блок параметров BIOS */

typedef struct _EBPB_ {
     unsigned sectsize;
     char clustsize;
     unsigned ressecs;
     char fatcnt;
     unsigned rootsize;
     unsigned totsecs;
     char media;
     unsigned fatsize;
     unsigned seccnt;
     unsigned headcnt;
```

```
     unsigned hiddensec_low;
     unsigned hiddensec_hi;
     unsigned long drvsecs;
} EBPB;

/* Загрузочная запись для MS-DOS 4.01 */

typedef struct _BOOT_ {
     char jmp[3];
     char oem[8];
     EBPB bpb;
     char drive;
     char reserved;
     char signature;
     unsigned volser_lo;
     unsigned volser_hi;
     char label[11];
     char fat_format[8];
     char boot_code[450];

} BOOT;

/* Время последнего обновления файла */

typedef struct _FTIME_ {
     unsigned sec : 5, min : 6, hour : 5;
} FTIME;

/* Дата последнего обновления файла */

typedef struct _FDATE_ {
     unsigned day : 5, month : 4, year : 7;
} FDATE;

/* Дескриптор файла в каталоге */

typedef struct _FITEM_ {
     char name[8];
     char ext[3];
     char attr;
     char reserved[10];
     FTIME time;
     FDATE date;
     unsigned cluster_nu;
     unsigned long size;
} FITEM;

/* Формат трека для GENERIC IOCTL */

typedef struct _TRK_LY_ {
     unsigned no;
     unsigned size;
} TRK_LY;

/* Параметры устройства для GENERIC IOCTL */

typedef struct _DPB_ {
```

```
    char spec;
    char devtype;
    unsigned devattr;
    unsigned numofcyl;
    char media_type;

    EBPB bpb;
    char reserved[6];

    unsigned trkcnt;
    TRK_LY trk[100];
} DPB;

/* Параметры для форматирования функцией GENERIC IOCTL */

typedef struct _DPB_FORMAT_ {
    char spec;
    unsigned head;
    unsigned track;
} DPB_FORMAT;

/* Параметры для чтения/записи функцией GENERIC IOCTL */

typedef struct _DPB_WR_ {

    char spec;
    unsigned head;
    unsigned track;
    unsigned sector;
    unsigned sectcnt;
    void _far *buffer;
} DPB_WR;

/* Идентификатор BIOS */

typedef struct _BIOS_ID_ {

    char date[8];
    unsigned reserve;
    char pc_type;
} BIOS_ID;

// Состояние мыши

typedef struct _MOUSE_STATE_ {

    unsigned bottoms;
    unsigned x;
    unsigned y;
} MOUSE_STATE;

typedef struct _SYSTIMER_ {

    char hour;
    char min;
```

```
    char sec;
    unsigned year;
    char month;
    char day;
    char daylight_savings;
} SYSTIMER;

#pragma pack()

void far *get_cvt(void); /* получить адрес
        векторной таблицы связи */
CVT  far *get_mcvt(void); /* получить адрес
        векторной таблицы связи */

MCB  far *get_fmcb(CVT far *); /* получить адрес первого MCB */
MCB  far *get_nmcb(MCB far *); /* получить адрес
        следующего MCB */

DDCB  far *get_fddcb(CVT far *); /* получить адрес
        первого DDCB */
DDCB  far *get_nddcb(DDCB far *); /* получить адрес
        следующего DDCB */
DDCB  far *get_ddcb(unsigned char); /* получить адрес
        DDCB для диска */

DFT  far *get_fdft(CVT far *); /* получить адрес первой DFT */
DFT  far *get_ndft(DFT far *); /* получить адрес
        следующей DFT */

BCB  far *get_fbcb(CVT far *); /* получить адрес первого BCB */
BCB  far *get_nbcb(BCB far *); /* получить адрес
        следующего BCB */

int get_exeh(EXE_HDR *,RELOC_TAB **, FILE *); /* прочитать
        заголовок EXE */

char unsigned pc_model(void); /* получить модель компьютера */
void disk_cfg(DISK_CONFIG*);  /* определить конфигурацию
        дисковой подсистемы */
DPT _far *get_dpt(void);      /* получить адрес DPT */
HDPT _far *get_hdp1(void);    /* получить адрес первой HDPT */,
HDPT _far *get_hdp2(void);    /* получить адрес второй HDPT */

BIOS_ID _far *getbiosi(void); /* получить адрес
        идентификатора BIOS */


int   ms_init(int *);           // Инициализация мыши
void  ms_on(void);              // Включение курсора
void  ms_off(void);             // Выключение курсора
void  ms_setcr(int, int);       // Установка курсора
int   ms_querp(MOUSE_STATE *, int); // Определение состояния
                          // мыши при нажатии на клавишу
void  ms_rangx(int xmin, int xmax); // Задать диапазон
                          // перемещений курсора по горизонтали
```

```
void  ms_rangy(int ymin, int ymax); // Задать диапазон
                                    // перемещений курсора
                                    // по вертикали
void  ms_gform(int xt, int yt, char _far *form); // определение
                                    // формы курсора
                                    // в графическом режиме
void ms_tform(int type, int mask1, int mask2); // определение
                                    // формы курсора
                                    // в текстовом режиме
MOUSE_STATE *ms_querm(MOUSE_STATE *state); // определение
                                    // относительного
                                    // перемещения в миках
void ms_seth(int mask, void (far *hand)());  // установка
                                    // драйвера событий

// Системные часы реального времени

#define RTC_GET_TIME    2
#define RTC_SET_TIME    3
#define RTC_GET_DATE    4
#define RTC_SET_DATE    5
#define RTC_SET_ALARM   6
#define RTC_CLEAR_ALARM 7

int timer(char, SYSTIMER *);  // работа с часами реального
                              // времени
void tm_delay(int);       // формирование задержки по таймеру
void tm_sound(int, int); // формирование тона заданной
                         // длительности с использованием
                         // таймера

void rnd_set(int); // инициализация генератора случайных чисел
int rnd_get(void); // получение случайного числа

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 aux_init(AUX_MODE *, int, int); // инициализация
                                    // асинхронного адаптера
void aux_stat(AUX_MODE *, int);     // определение режима
                                    // асинхронного адаптера
void aux_outp(char, int);           // вывод символа
                                    // в асинхронный адаптер
char aux_inp(int);                  // ввод символа
                                    // из асинхронного адаптера

// Прототипы функций для работы с расширенной памятью.

unsigned XMM_Installed();

long  XMM_Version(void);
long  XMM_RequestHMA(unsigned);
long  XMM_ReleaseHMA(void);
long  XMM_GlobalEnableA20(void);
long  XMM_GlobalDisableA20(void);
long  XMM_EnableA20(void);
long  XMM_DisableA20(void);
long  XMM_QueryA20(void);
long  XMM_QueryLargestFree(void);
long  XMM_QueryTotalFree(void);
long  XMM_AllocateExtended(unsigned);
long  XMM_FreeExtended(unsigned);
long  XMM_MoveExtended(struct XMM_Move *);
long  XMM_LockExtended(unsigned);
long  XMM_UnLockExtended(unsigned);
long  XMM_GetHandleLength(unsigned);
long  XMM_GetHandleInfo(unsigned);
long  XMM_ReallocateExtended(unsigned,unsigned);
long  XMM_RequestUMB(unsigned);
long  XMM_ReleaseUMB(unsigned);

struct   XMM_Move {
     unsigned long  Length;
     unsigned short SourceHandle;
     unsigned long  SourceOffset;
     unsigned short DestHandle;
     unsigned long  DestOffset;
};

// Прототипы функций для работы с дополнительной памятью.

int ems_init(void);
int ems_stat(void);
int ems_fram(unsigned *);
int ems_page(unsigned *, unsigned *);
int ems_open(int, int *);
int ems_clos(int *);
int ems_map(int, int, int);
int ems_ver(char *);
```

Приложение 12

КОМАНДЫ СОПРОЦЕССОРОВ 8087/80287/80387

Загрузка данных в стек

FLD src	Загрузка вещественного числа st(0) <- src (mem32/mem64/mem80)
FILD src	Загрузка целого числа st(0) <- src (mem16/mem32/mem64)
FBLD src	Загрузка десятичного числа st(0) <- src (mem80)

Загрузка констант

FLDZ	Загрузка нуля st(0) <- 0.0
FLD1	Загрузка единицы st(0) <- 1.0
FLDPI	Загрузка числа Pi st(0) <- Pi
FLDL2T	Загрузка log2(10) st(0) <- log2(10)
FLDL2E	Загрузка log2(e) st(0) <- log2(e)
FLDLG2	Загрузка log10(2) st(0) <- log10(2)
FLDLN2	Загрузка loge(2) st(0) <- loge(2)

Приложение 10

СОДЕРЖИМОЕ ФАЙЛА sysp.h

```
/* SYSP.H - include-файл для примеров, приведенных в книге */

/**
*.Name       FP_MAKE
*
*.Title      Макро для составления FAR-указателя
*
*.Descr      Макро составляет FAR-указатель, пользуясь
*            значениями сегмента и смещения
*
*.Params     FP_MAKE(seg,off)
*               seg - сегмент;
*               off - смещение
*
*.Return     FAR-указатель seg:off
**/

#define FP_MAKE(seg,off) ((void far *)  \
     ((((unsigned long) (unsigned)(seg)) << 16L) |   \
     ((unsigned long) (unsigned) (off))))

/* Структура векторной таблицы связи DOS */

#pragma pack(1)

typedef struct _CVT_ {
     unsigned mcb_seg;
     void far *dev_cb;
     void far *file_tab;
     void far *clock_dr;
     void far *con_dr;
     unsigned max_btbl;
     void far *disk_buf;
     void far *drv_info;
     void far *fcb_tabl;
     unsigned fcb_size;
     unsigned char num_bdev;
     unsigned char lastdriv;
} CVT;

/* Блок управления памятью MCB */

typedef struct _MCB_ {
     unsigned char type;
     unsigned owner;
     unsigned size;
     char reserve[11];
} MCB;
```

Запись данных

FST dest — Запись вещественного числа
dest <- st(0) (mem32/mem64)

FSTP dest — Запись вещественного числа с извлечением его из стека численных регистров
dest <- st(0) (mem32/mem64/mem80)

FIST dest — Запись целого числа
dest <- st(0) (mem32/mem64)

FISTP dest — Запись целого числа с извлечением его из стека численных регистров
dest <- st(0) (mem16/mem32/mem64)

FBST dest — Запись десятичного числа
dest <- st(0) (mem80)

FBSTP dest — Запись десятичного числа с извлечением его из стека численных регистров
dest <- st(0) (mem80)

Сравнение

FCOM — Сравнение вещественных чисел
Установка флагов по результатам операции
st(0) - st(1)

FCOM op — Сравнение вещественных чисел
Установка флагов по результатам операции
st(0) - op (mem32/mem64)

FCOMP op — Сравнение вещественных чисел с извлечением из стека
Установка флагов по результатам операции
st(0) - op (mem32/mem64)

FCOMPP — Сравнение вещественных чисел с двойным извлечением из стека численных регистров
Установка флагов по результатам операции
st(0) - st(1)

FICOM op	Сравнение целых чисел Установка флагов по результатам операции st(0) - op (mem16/mem32)
FICOMP op	Сравнение целых чисел с извлечением из стека численных регистров Установка флагов по результатам операции st(0) - op (mem16/mem32)
FTST	Сравнение с нулем Сравнивается st(0) и 0.0
FUCOM st(i)	Неупорядоченное сравнение (только 80486) Содержимое st(0) сравнивается с st(i)
FUCOMP st(i)	Неупорядоченное сравнение (только 80486) с извлечением из стека численных регистров Содержимое st(0) сравнивается с st(i)
FUCOMPP st(i)	Неупорядоченное сравнение (только 80486) с двойным извлечением из стека численных регистров Содержимое st(0) сравнивается с st(i)
FXAM	Проверка верхушки стека По состоянию st(0) устанавливаются коды условий

Арифметические команды

FADD	Сложение вещественных чисел st(0) <- st(0) + st(1)
FADD src	Сложение вещественных чисел st(0) <- st(0) + src (mem32/mem64)
FADD st(i),st	Сложение вещественных чисел st(i) <- st(i) + st(1)
FADDP st(i),st	Сложение вещественных чисел с извлечением из стека численных регистров st(i) <- st(i) + st(1)

FIADD src	Сложение целых чисел st(0) <- st(0) + src (mem16/mem32)
FSUB	Вычитание вещественных чисел st(0) <- st(0) - st(1)
FSUB src	Вычитание вещественных чисел st(0) <- st(0) - src
FSUB st(i),st	Вычитание вещественных чисел st(i) <- st(i) - st(1)
FSUBP st(i),st	Вычитание вещественных чисел с извлечением из стека численных регистров st(i) <- st(i) - st(1)
FSUBR st(i),st	Вычитание вещественных чисел обратное st(0) <- st(i) - st(0)
FSUBRP st(i),st	Вычитание вещественных чисел обратное с извлечением из стека численных регистров st(0) <- st(i) - st(0)
FISUB src	Вычитание целых чисел st(0) <- st(0) - src (mem16/mem32)
FISUBR src	Вычитание целых чисел обратное st(0) <- src (mem16/mem32) - st(0)
FMUL	Умножение вещественных чисел st(0) <- st(0) * st(1)
FMUL st(i)	Умножение вещественных чисел st(0) <- st(0) * st(i)
FMUL st(i),st	Умножение вещественных чисел st(i) <- st(0) * st(i)
FMULP st(i),st	Умножение вещественных чисел с извлечением из стека численных регистров st(i) <- st(0) * st(i)

FIMUL src	Умножение целых чисел st(0) <- st(0) * src (mem16/mem32)
FDIV	Деление вещественных чисел st(0) <- st(0) / st(1)
FDIV st(i)	Деление вещественных чисел st(0) <- st(0) / st(i)
FDIV st(i), st	Деление вещественных чисел st(i) <- st(0) / st(i)
FDIVP st(i), st	Деление вещественных чисел с извлечением из стека численных регистров st(i) <- st(0) / st(i)
FIDIV src	Деление целых чисел st(0) <- st(0) / src (mem16/mem32)
FDIVR st(i), st	Деление вещественных чисел обратное st(0) <- st(i) / st(0)
FDIVRP st(i), st	Деление вещественных чисел обратное с извлечением из стека численных регистров st(0) <- st(i) / st(0)
FIDIVR src	Деление целых чисел обратное st(0) <- src (mem16/mem32) / st(0)
FSQRT	Извлечение квадратного корня st(0) <- × st(0)
FSCALE	Масштабирование на степень числа 2 st(0) <- 2 | st(0)
FXTRACT	Извлечение экспоненты st(0) <- значение экспоненты st(0)
FPREM	Вычисление частичного остатка st(0) <- st(0) MOD st(1)

FPREM1 — Вычисление частичного остатка в стандарте IEEE, только для 80486
st(0) <- st(0) MOD st(1)

FRNDINT — Округление до ближайшего целого
st(0) <- INT(st(0))

FABS — Вычисление абсолютного значения
st(0) <- ABS(st(0))

FCHS — Изменение знака
st(0) <- -st(0)

Трансцендентные команды

FCOS — Вычисление косинуса
st(0) <- COS(st(0))

FPTAN — Вычисление частичного тангенса
st(0) <- TAN(st(0))

FPATAN — Вычисление частичного арктангенса
st(0) <- ATAN(st(0))

FSIN — Вычисление синуса
st(0) <- SIN(st(0))

FSINCOS — Вычисление синуса и косинуса
st(1) <- SIN(st(0))
st(0) <- COS(st(0))

F2XM1 — Вычисление выражения 2x-1
st(0) <- 2st(0)-1

FYL2X — Вычисление Y * log2(X)
Y = st(0), X = st(1)
st(0) <- Y * log2(X)

FYL2XP1 — Вычисление Y * log2(X+1)
Y = st(0), X = st(1)
st(0) <- Y * log2(X+1)

Управляющие команды

FINIT	Инициализация арифметического сопроцессора
FSTSW AX	Запись слова состояния AX <- MSW
FSTSW dest	Запись слова состояния dest (mem16) <- MSW
FLDCW src	Загрузка управляющего слова CW <- src (mem16)
FSTCW dest	Запись управляющего слова dest (mem16) <- CW
FCLEX	Сброс флагов особых случаев
FSTENV dest	Запись содержимого всех регистров сопроцессора, кроме численных
FLDENV src	Загрузка содержимого всех регистров сопроцессора, кроме численных
FSAVE dest	Запись содержимого всех регистров сопроцессора
FRSTOR src	Загрузка содержимого всех регистров сопроцессора
FINCSTP	Увеличение указателя стека численных регистров на 1 st(6) <- st(5) st(5) <- st(4) st(4) <- st(3) st(3) <- st(2) st(2) <- st(1) st(1) <- st(0) st(0) <- пустой
FDECSTP	Уменьшение указателя стека численных регистров на 1 st(0) <- st(1) st(1) <- st(2) st(2) <- st(3) st(3) <- st(4) st(4) <- st(5) st(5) <- st(6)

st(6) <- st(7)
st(7) <- пустой

FFREE — Освобождение регистра st(i)

Команда помечает численный регистр st(i) как неинициализированный, записывая в соответствующее поле регистра тегов значение 11

FNOP — Холостая команда

st(0) <- st(0)

WAIT/FWAIT — Синхронизация центрального процессора и арифметического сопроцессора

Центральный процессор будет находиться в состоянии ожидания до тех пор, пока сопроцессор не завершит выполнение текущей команды

FSETPM — Установить защищенный режим работы

Процессор переходит в защищенный режим работы, возврат в реальный режим возможен только при сбросе сопроцессора

Приложение 13

ФОРМАТЫ ДАННЫХ СОПРОЦЕССОРА

Одинарная точность

1 бит 8 бит 23 бита

Двойная точность

1 бит 11 бит 52 бита

Расширенная точность

1 бит 15 бит 64 бита

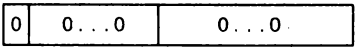
В любом представлении старший бит "Зн" определяет знак вещественного числа:

0 - положительное число;

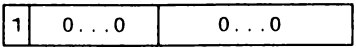
1 - отрицательное число.

Возможные значения вещественных чисел:

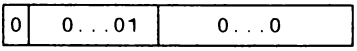
Положительный нуль



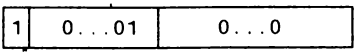
Отрицательный нуль



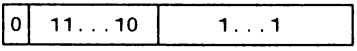
Наименьшее положительное число



Наибольшее отрицательное число

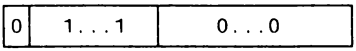


Наибольшее положительное число

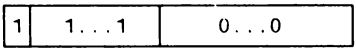


Наименьшее отрицательное число

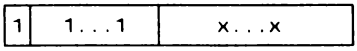
Положительная бесконечность



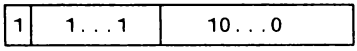
Отрицательная бесконечность



Нечисло

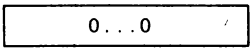


Неопределенность

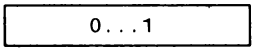


Возможные значения целых чисел:

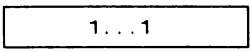
Нуль



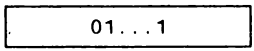
Наименьшее положительное число



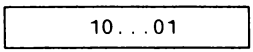
Наибольшее отрицательное число



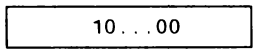
Наибольшее положительное число



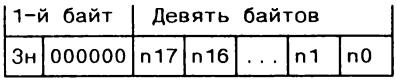
Наименьшее отрицательное число



Неопределенность



Упакованное десятичное число:



ОГЛАВЛЕНИЕ

ДИАЛОГ-МИФИ

АО "ДИАЛОГ-МИФИ" выпускает литературу по программированию и вычислительной технике, рассчитанную на широкий круг пользователей персональных компьютеров.
В серии **"Библиотека системного программиста"** вышли:

Том 1
ОПЕРАЦИОННАЯ СИСТЕМА MS-DOS, в 3-х книгах
А.В. Фролов, Г.В. Фролов

От аналогичных изданий отличается более глубоким изложением материала. Описываются детали, которые часто остаются скрытыми даже для опытного программиста: внутренняя структура и организация работы MS-DOS, недокументированные прерывания. Разбираются профессиональные приемы работы. Приводятся подробно комментируемые исходные тексты программ.

Том 2
АППАРАТНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРА IBM PC, в 2 частях
А.В. Фролов, Г.В. Фролов

Вы держите сейчас в руках.

Том 3
ПРОГРАММИРОВАНИЕ ВИДЕОАДАПТЕРОВ CGA, EGA и VGA
А.В. Фролов, Г.В. Фролов

Содержит подробное описание архитектуры и программирования видеоадаптеров CGA/EGA/VGA. Описано использование регистров видеоадаптера, стандартные режимы работы и структура видеопамяти в них. Приведен обзор прерываний и функций BIOS для обслуживания видеоадаптеров. Приведены основные графические функции стандартных библиотек трансляторов Microsoft QC 2.5 и C 6.0. Книга содержит примеры программ, составленных на языках ассемблера и Си.

ГОТОВИТСЯ К ПЕЧАТИ:

Том 4
ПРОГРАММИРОВАНИЕ МОДЕМОВ
А.В. Фролов, Г.В. Фролов

Посвящен средствам, используемым для связи персональных компьютеров друг с другом. Приведены два типа соединений: через асинхронный порт и связь с использованием модема. Рассмотрены вопросы программирования асинхронного последовательного адаптера, описаны его порты и режимы, средства BIOS для работы с адаптером, а также соответствующие функции стандартных библиотек компиляторо Си. Кратко рассмотрены программы Norton Commander и FastWire. Книга содержит инструкции по установке модема на компьютере, описывает режимы работы модемов.

К каждому тому дополнительно выпущены дискеты с текстами программ и справочной информацией.

ПРОГРАММИРОВАНИЕ В MICROSOFT WINDOWS, в 2 частях
С.А. Гладков, Г.В. Фролов

Учебно-справочное пособие по созданию программ ("приложений"), работающих под управлением операционной оболочки MS-Windows 2.x, 3.x. Рассматривается положение MS-Windows среди системного программного обеспечения, ее внутренняя структура. Обсуждаются вопросы установки среды программирования, механизмы взаимодействия приложения с операционной оболочкой, алгоритм создания программы. Раскрываются аспекты программирования в Windows: структура программы, работа с экраном, клавиатурой, мышью, графика, шрифты и пр. Предметный указатель позволяет использовать книгу как справочник функций MS-Windows. Дополнительно можно приобрести дискету с примерами программ.

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АВТОЛИСП В СИСТЕМЕ САПР АВТОКАД
С.А. Гладков

Учебно-справочное пособие по программированию. Описываются структура языка, распределение памяти и обращение к графической базе данных Автокада, даются методика написания и примеры программ. Рассмотрены вопросы написания макроопределений, меню Автокада и настройки рабочей среды. Ориентировано на пользователей персональных компьютеров, работающих в области САПР.

ПРАКТИЧЕСКАЯ РАБОТА НА КОМПЬЮТЕРАХ СЕМЕЙСТВА IBM PC В ОПЕРАЦИОННОЙ СРЕДЕ MS-DOS 4.01
Ю.Я. Максимов, С.В. Осипов, О.С. Симоненков

Пособие содержит краткое описание операционной системы MS-DOS 4.01 и практических приемов работы в ней. Главы, содержащие сведения о персональных компьютерах (ПК), файловой системе и основных командах MS-DOS, а также об операционной оболочке MS-DOS Shell, полезны как для начинающих, так и для более опытных пользователей. Главы, посвященные операционным системам ПК, структуре и функциям операционной системы MS-DOS и командным файлам.

С для PC. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С ДЛЯ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ
А. Григорьев

Вып. 1. Связь С-программ с операционной системой
Вып. 2. Время и звук
Вып. 3. Стандартный ввод/вывод и прерывания
Вып. 4. Графика на экране I
Вып. 5. Графика на экране II
Вып. 6. Текст и графика на принтере

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МУЛЬТИТРАНСПЬЮТЕРНЫХ СИСТЕМ
Б.Ю. Сырков, С.В. Матвеев

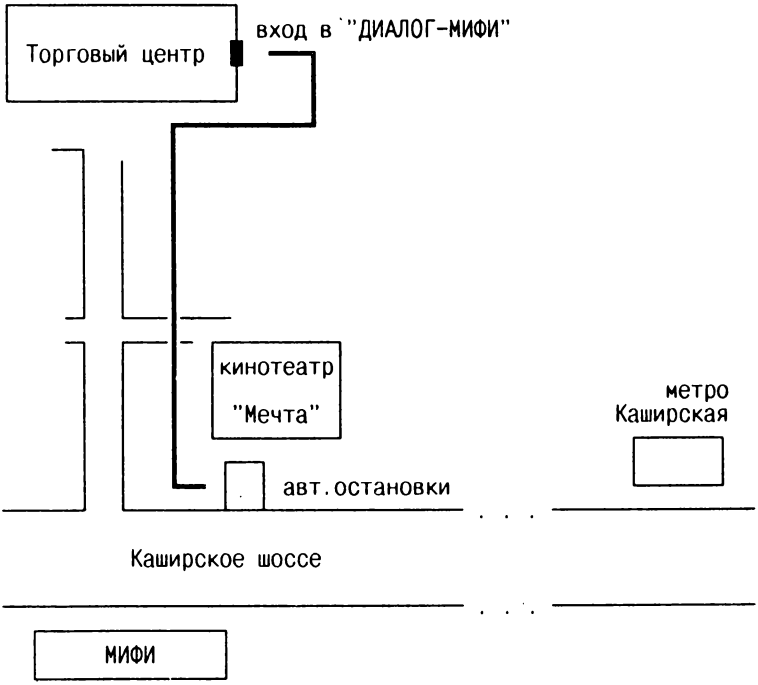
Рассмотрены основы функционирования и использования программного обеспечения мультитранспьютерных систем. Приведены оценки

производительности транспьютера; описан язык программирования высокого уровня Оккам-2, распределенная операционная система Гелиос; особое внимание уделено средствам создания параллельных программ для мультитранспьютеров в среде Гелиос; рассмотрено программирование мультитранспьютерных систем с использованием последовательных языков высокого уровня (Си, Паскаль, Фортран, Модула-2); изложение иллюстрировано примерами.

**Эти и другие издания "ДИАЛОГ-МИФИ" можно приобрести по адресу: Москва, ул. Москворечье, 31, корп. 2.
Проезд: м. Каширская, авт. 95, 117, 162, 192, 275, 709, 740 до ост. "МИФИ" (затем спрашивать "Торговый центр")**

УСЛОВИЯ ОПЛАТЫ И СКИДКИ ДЛЯ ОПТОВЫХ ПОКУПАТЕЛЕЙ ЗАВИСЯТ ОТ ОБЪЕМА ЗАКУПАЕМЫХ ПАРТИЙ И СОГЛАСУЮТСЯ В КАЖДОМ СЛУЧАЕ ОТДЕЛЬНО.

ПРИГЛАШАЕМ РАСПРОСТРАНИТЕЛЕЙ ПЕЧАТНОЙ ПРОДУКЦИИ. ФОРМА ОПЛАТЫ ПО ДОГОВОРЕННОСТИ. С ПРЕДЛОЖЕНИЯМИ ОБРАЩАТЬСЯ ПИСЬМЕННО ПО УКАЗАННОМУ ВЫШЕ АДРЕСУ ИЛИ ПО ТЕЛЕФОНУ (095) 320-43-77, ФАКС (095) 3243055.



ДИАЛОГ-МИФИ

АО "ДИАЛОГ-МИФИ", официальный дистрибутор фирмы Symantec Corporation, предлагает широкий набор лицензионных программных продуктов:

• *Интегрированные пакеты* • *Профессиональные компиляторы* • *Утилиты* • *Текстовые процессоры* •

Обеспечивается *сопровождение* **программных продуктов. Покупатели становятся** *зарегистрированными пользователями* **фирмы Symantec и им предоставляются** *скидки* **при последующих приобретениях программных продуктов.**

SYMANTEC.

Акционерное общество "ДИАЛОГ-МИФИ". 115409 Москва, ул. Москворечье, 31, корп. 2 Тел.: 320-32-11, 324-43-44. Факс: (095) 3243055

ДИАЛОГ-МИФИ

АО "ДИАЛОГ-МИФИ", официальный дистрибутор фирмы Borland International, предлагает широкий набор лицензионных программных продуктов:

• *Языки программирования* • *Базы данных* • *Электронные таблицы* • *Инструментальные средства* • *Интегрированные пакеты* •

Обеспечивается *сопровождение* программных продуктов. Покупатели становятся *зарегистрированными пользователями* фирмы Borland и им предоставляются *скидки* при последующих приобретениях программных продуктов.

B O R L A N D

Акционерное общество "ДИАЛОГ-МИФИ".
115409 Москва,
ул. Москворечье,
31, корп. 2
Тел.: 320-32-11,
324-43-44.
Факс: (095) 3243055

ДИАЛОГ-МИФИ
SYMANTEC.

Акционерное общество "ДИАЛОГ-МИФИ", официальный дистрибутор *Symantec Corporation*, предлагает русскоязычную версию Time Line 4.0 - одного из самых популярных на сегодняшний день программных пакетов программ для управления проектами на компьютерах IBM PC.

Принято считать, что при нынешнем соотношении цен проще нанять одного-двух человек, чем покупать программу за десятки тысяч рублей. Это глубокое заблуждение.
На самом деле для улучшения работы следует вначале повысить эффективность использования имеющихся ресурсов - как людей, так и оборудования, не исключая и руководителя, менеджера.

Более 25% менеджеров во всем мире используют для управления своими проектами именно Time Line. Наиболее широкое применение в США этот пакет нашел в строительстве и машиностроении: с использованием Time Line создавались автомобили, небоскребы, скоростные магистрали и многое другое. Одним из крупнейших заказчиков Symantec на Time Line является NASA (Национальное Управление по Аэронавтике и Исследованиям Космического Пространства). В нашей стране первыми пользователями Time Line стали также строительные организации, институты атомной промышленности, КБ Туполева и др.

Time Line предлагает Вам не только инструментальные средства решения Ваших проблем, но и новую концепцию управления, позволяющую сделать выполнение проекта более экономичным, гибким. Известно, что проекты, реализованные с помощью Time Line, обходились как минимум на 20% дешевле.

TIME LINE

Технология управления, предлагаемая Time Line, дает возможность представить картину проекта как единое целое, четко установить связь между его элементами.

Краткие технические характеристики Time Line:

- Планирование от объема работ
- Регулирование загрузки ресурсов
- Планирование (создание плана) и управление (текущий контроль по принятому плану)
- 1000 задач и 300 ресурсов в расписании работ
- Возможность анализа "что/если"
- Одновременная работа нескольких пользователей в сети
- Широкий спектр различных отчетов
- Экспорт/импорт из/в dBASE, Excel, Lotus 1-2-3 (Works), ASCII и др.

Спешите приобрести Time Line, - это хорошая возможность для Вас!

Мы привели, конечно, только самую краткую информацию; более подробно Вы можете узнать о Time Line по телефонам: 320-32-11, 320-43-66, 320-71-66 или отправить факс по номеру: 324-30-55

Наш адрес: 115409 Москва, ул. Москворечье, 31, корп. 2, АО "ДИАЛОГ-МИФИ"

"ДИАЛОГ-МИФИ" ПРЕДЛАГАЕТ

ПРОФЕССИОНАЛЬНЫЕ ГРАФИЧЕСКИЕ РАБОЧИЕ СТАНЦИИ ДЛЯ САПРа

Цветная графическая рабочая станция HP 9000 Series 300:

16.7 - 50 MHz 68030 процессор, 68882 сопроцессор, 8-32 Mb ОЗУ, HP-IB, RS-232, Centronics, LAN интерфейсы, высокоскоростной HP-IB дисковый интерфейс. Операционная система HP-UX включает: Execution and Programming Environment, C Compiler, 680x0 Assembler, Symbolic Debugger, X-Windows, Starbase Graphics Library and OSF/Motif Interface. 16" или 19" цветной видеомонитор с высоким разрешением и видеоинтерфейс. 330/660 Megabyte Hard Disc, 3.5" - Floppy Disc, A3 или A4 дигитайзер. Дополнительно: 67 или 133 Mb 1/4" Tape Streamer. Дополнительно: любой HP Plotter формата от A4 до A0. Дополнительно: любой HP Printer, включая лазерные, цветные, струйные и т. д.

САПР для машиностроения фирмы HP

Система САПР Mechanical Engineering: ME 10 для конструкторов и чертежников. Включает в себя также модуль параметризации, систему доступа к базам данных, модуль макроопределений, систему составления спецификаций и инструмент для изометрического представления чертежа. Реляционная СУБД SQL на сервере DXF и IGES транслятор. Возможность подключения к ЧПУ. Система САПР Mechanical Engineering: ME 30 для твердотельного (Solid) конструирования. Включает в себя все возможности Mechanical Engineering ME 10, а также: 3-х мерное (Solid) конструирование с возможным сопряжением с системами: APT-AC, COMPACT-II, GNC, ANSYS, FEMGEN, PATRAN, SUPERTAB (I-DEAS), IGES 3D, VDA-FS.

САПР для машиностроения фирмы SDRC

I-DEAS - это высоко технологичная система для проектирования в области машиностроения. Включает в себя: 3-х мерное твердотельное (Solid) моделирование и 2-х мерное конструирование. Геометрическое моделирование с NURBS, моделирование сборки, модуль оптимизации. Расчеты методом конечных элементов с автоматическим и ручным разбиением. Проектирование и оптимизация, статические, динамические и тепловые расчеты. Частотный анализ. ЧПУ программирование и т.д.

ДИАЛОГ-МИФИ

115409 Москва,
ул. Москворечье,
д. 31, корп. 2,
Тел. 320-31-33
Факс (095) 3243055

ДИАЛОГ·МИФИ

УЧЕБНЫЙ ЦЕНТР "ДИАЛОГ-МИФИ"

Учебный центр проводит обучение работе на персональных компьютерах с использованием новейшего программного обеспечения. Курсы **авторизованы фирмами Borland International Inc., Symantec Corporation** и **Autodesk Inc.** За три года Учебным центром подготовлено более 1300 слушателей из стран Европы, Азии и Африки.

- На курсах можно получить как **начальные навыки** работы с персональным компьютером, так и **углубленные знания** в области программирования и использования фирменных программных продуктов.
- **Продолжительность** занятий две недели с отрывом от производства (занятия - пять раз в неделю). Каждый день занятий - это трехчасовая лекция и пять часов индивидуальной практической работы на персональных компьютерах IBM PC под руководством преподавателей Учебного центра и МИФИ (1 преподаватель - 4 слушателя).
- По каждому курсу имеется **оригинальное пособие**.
- Иногородним предоставляется **жилье**.
- Зачисление на курсы слушателей, направляемых организацией, проводится по гарантийным письмам.
- Окончившим курсы выдается **удостоверение-сертификат**.
- Учебный центр организует и проводит занятия **на базе заказчика**.
- Для желающих проводятся платные **консультации**, принимаются платные **экзамены** по преподаваемым курсам с выдачей удостоверения.
- Можно приобрести **литературу** по преподаваемым курсам.

115409 Москва, ул. Москворечье, 31, корп. 2
Телефоны: 320-30-88, 324-83-56.
Телефакс: (095) 3243055