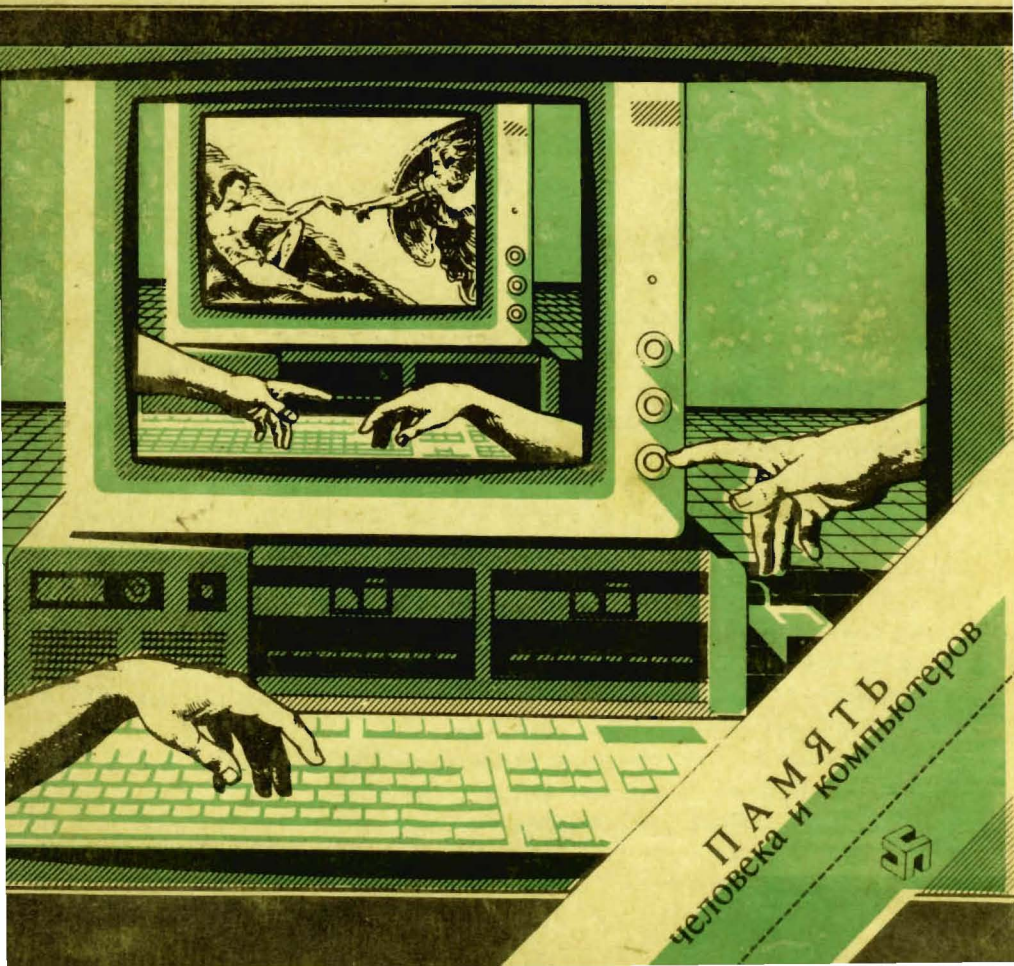
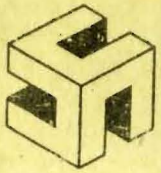
СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ
на ассемблере
для
IBM-СОВМЕСТИМЫХ
персональных компьютеров



СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ
на ассемблере
для
IBM-СОВМЕСТИМЫХ
персональных компьютеров



МИП "Память"
совместно с Российско-американским университетом
готовит к выпуску книги:

1. Циолковский К.Э. — *Ум и страсти. Воля вселенной. Неизвестные разумные силы.*
2. Циолковский К.Э. — *Общественная организация человечества.*
3. Циолковский К.Э. — *Любовь к самому себе или истинное себялюбие.*
4. Циолковский К.Э. — *Прошедшее земли.*
5. Даувальдер Валерия Ф. — *Историческая реальность Христа.*
6. Даувальдер Валерия Ф. — *Повесть кота Тараса.*
7. Дегтярев Е.К. — *Тенденции развития вычислительной техники.*

Предлагаем

1. Архитектура, протоколы и тестирование открытых информационных сетей. Толковый словарь / В.Ф. Баумгарт, С.П. Волкова, А.В.Гнедовский и др.; Под ред. Э.А.Якубайтиса. - 1989.- 192 с.

2. Рыбкин Е.Н., Юдин А.Ю. Программные средства ПЭВМ: операционная система MS-DOS. ,1991. - 112 сл.: ил.

3. Прокофьев Б.П., Сухарев Н.Н., Храмов Ю.Е. Графические средства Turbo C и Turbo C ++ / Под ред. Г.В.Генса, Ю.Е.Храмова.- 1992. - 160 с.

4. Смородинский А.В., Моисеев В.А., Стовманенко Л.М., Посудина О.Ю. АВТОКАД ДЛЯ НОВИЧКОВ И ПРОФЕССИОНАЛОВ. Книга 1. Начинаем освоение системы. -, 1991. - 144 с.: ил.

5. Керниган Б., Ритчи Д. Язык программирования Си: Пер. с англ. / Под ред. и с предисл. Вс.С.Штаркмана. - 2-е изд., перераб. и доп.- 1992.-272 с.: ил.

6. Джордейн Р. Справочник программиста персональных компьютеров типа IBM PC, XT и AT: Пер. с англ. / Предисл. Н.В.Гайского. 1991. - 544 с.: ил.

7. Романовская Л.М. и др. Программирование в среде Си для ПЭВМ ЕС / Л.М.Романовская, Т.В.Русс, С.Г.Свитковский. , 1991. - 352 с.: ил.

8. Фигурнов В.Э. Использование персональных компьютеров IBM PC. В мягкой обложке

А.В. Богословский

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

НА АССЕМБЛЕРЕ

ДЛЯ

IBM-СОВМЕСТИМЫХ

ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Москва

-1992-

УДК 519.68•681.3.06

Богословский А.В.
Системное программирование на Ассемблере для IBM-совместимых персональных компьютеров.
©М.: Память., 1992 г.

ISBN 5-87140-056-6
Пособие предназначено для программистов, знакомых с ассемблером, желающих углубить свои знания операционной системы MS-DOS. Приведены программы: определение типа видеоадаптера и работа с ним; чтение таблицы разделов жесткого диска; нестандартного форматирования дорожки дискеты с целью защиты от копирования; использования HOT KEY в резидентных программах; драйвера и его отладки.

Рецензент: Малютин Э.А.

ISBN 5-87140-056-6

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель этой книги - дать сведения по некоторым, не слишком хорошо известным вопросам программирования для компьютеров семейства IBM PC и совместимым с ними. Содержание книги основано на курсах лекций, прочитанных автором в учебных центрах СП "ИНТЕРКВАДРО" и "ЛАНИТ". В дальнейшем предполагается выпустить серию из нескольких книг посвященных следующим вопросам:

1. Недокументированные функции и управляющие блоки DOSa.

2. Программирование видеоадаптеров.

3. Программные драйверы.

4. Структурное программирование на языке Ассемблера (с использованием макро-определений пользователя).

В этой серии максимально полно будут рассматриваться вопросы, отсутствующие в документации. Возможности, с которыми легко ознакомиться по документации и существующей литературе, специально рассматриваться не будут, а будут упоминаться в изложении только в тех случаях, когда это необходимо для ясности и полноты изложения. Все сказанное относится и к этой книге. Все примеры отлажены и проверены при работе под управлением MS-DOS (зарегистрированная торговая марка фирмы MICROSOFT) версий от 3.30 до 5.0. Примеры оттранслированы с помощью транслятора MASM (зарегистрированная торговая марка фирмы MICROSOFT) версии 5.10.

Автор заранее благодарен всем, кто возьмет на себя труд сообщить свое мнение о содержании этой книги, высказать свои замечания и предложения.

BIOS

BIOS - базовая система ввода/вывода (Basic Input Output System) - программа, управляющая работой всех стандартных устройств компьютера на низком уровне, начальной инициализацией и проверкой всех устройств компьютера. Под "НИЗКИМ"

уровнем здесь подразумевается уровень при котором для осуществления ввода/вывода используются функции и сервис BIOSa и, следовательно, программа остается переносимой, то есть, ее можно перенести с одного компьютера на другой, не внося изменений в тело программы. Иногда еще говорят, что программа в этом случае является "ЗАКОННОЙ". BIOS также осуществляет обработку внешних прерываний. Весь ввод/вывод на физическом уровне, с учетом особенностей конкретных устройств, производится программой BIOSa. После проверки и инициализации устройств BIOS пытается загрузить операционную систему с магнитного диска (сначала с устройства A: затем с C:) и, если попытка удалась, передает ей (операционной системе) управление. Если загрузить систему не удается, то делается попытка загрузить из ПЗУ встроенный интерпретатор BASICa. Если, по каким-либо причинам, например из-за того, что этого интерпретатора нет, как это обычно бывает в машинах класса AT и выше, это тоже оказывается невозможным, то выдается сообщение об ошибке интерпретатора и остановке компьютера, после чего он останавливается.

При использовании сервиса BIOSa программист должен самостоятельно следить за корректным использованием ресурсов системы. Использование сервиса BIOSa позволяет программисту получить доступ практически ко всем ресурсам машины, однако требует большой точности от программиста. Кроме того, пользование сервисом BIOSa часто сложнее, чем пользование возможностями и сервисом DOSa. Особенно эта разница заметна при сравнении дисковых функций BIOSa и DOSa. Программа BIOSa помещена в ПЗУ, в старшие адреса памяти (сегмент 0F000h, смещение зависит от типа машины и версии BIOSa), и получает управление сразу при включении питания, а также при перезагрузке компьютера.

Помимо основного BIOSa, управляющего работой компьютера, может быть установлен и добавочный, управляющий работой отдельных устройств. Для добавочного BIOSa отводится адресное пространство начиная с адреса 0C000h:0000. По этим адресам расположены, как правило, BIOS видеоадаптера и BIOS жесткого диска. Это связано с многообразием типов видеоадаптеров и адаптеров жестких дисков. Нет смысла помещать их в основной BIOS.

Сервис BIOSa

Сервис BIOSa вызывается программистом через соответствующие программные прерывания (команда процессора INT) и их функции. Для сервиса BIOSa зарезервированы номера прерываний от 10h до 1Fh. Каждое прерывание отвечает за работу какого либо устройства или группы однотипных устройств. Номер нужной функции заносится перед выдачей команды INT в регистр AH. Остальные регистры используются для передачи параметров и получения результатов работы функции. Сервис BIOSa для графических адаптеров вызывается через функции прерывания 10h, для дисковых устройств через прерывание 13h, для клавиатуры 16h, а для принтера 17h. Сервис BIOSa достаточно хорошо описан в литературе [2,3,4], поэтому, всех интересующихся этим, автор отсылает к этим книгам, а также, к широко распространенной в нашей стране программе "Tech help!" (The Electronic Technical Reference Manual фирмы Flambeaux Software).

Графические адаптеры

Первыми видеоадаптерами, примененными в IBM-совместимых персональных компьютерах, были MDA и CGA (Monochrome Display Adapter и Color Graphics Adapter), которые в настоящее время практически не применяются. Знакогенератор для этих адаптеров зашит в ПЗУ (Постоянное Запоминающее Устройство), поэтому в текстовом режиме отсутствовала возможность программного создания своих фонтов (ФОНТ - набор матриц знакогенератора - происходит от американского варианта FONT английского слова FOUNT, означающего комплект шрифта). В настоящее время эти адаптеры вытеснены существенно более совершенными. На сегодня фактическим стандартом в большинстве случаев является адаптер EGA (Enchanced Graphics Adapter) для цветного и HGC (Hercules Graphics Card) для монохромного адаптеров. Дальнейшим развитием цветного адаптера является VGA (Video Graphics Array), спроектированный для использования в машинах с 16-ти битовой шиной и, следовательно, более производительный. Кроме того VGA

содержит, как правило, значительно больший экранный буфер, что позволило улучшить разрешающую способность и увеличить количество цветов в палитре. Сейчас фирмой IBM разработан новый видеоадаптер - XGA (eXtended Graphics Array) с существенно лучшими возможностями, как по разрешению, так и по набору цветов. Кроме того встречаются видеоадаптеры MCGA (Multi Color Graphics Array) и, довольно редко, PGA (Professional Graphics Adapter).

При программировании графических адаптеров часто приходится пользоваться "**незаконными**" способами видеовывода, такими как прямой вывод в видеопамять, непосредственное программирование портов и так далее. Так приходится делать потому, что сервис BIOSa работает слишком медленно. Этими приемами пользуются многие популярные программы, например - Norton Commander, XTREE, компиляторы фирмы BORLAND и другие.

Программирование видеоадаптеров

Программирование всех видеоадаптеров в текстовом режиме практически одинаково, поэтому описываться будет в одном разделе, с указанием различий для конкретных адаптеров. Как для текстового, так и для графических режимов подробно будет рассматриваться только использование так называемого "**НЕЗАКОННОГО**" метода программирования, с использованием прямого доступа в видеопамять и непосредственного программирования портов ввода/вывода. При такой организации видеовывода программист должен самостоятельно решать следующие вопросы:

1. Определение типа видеоадаптера.
2. Определение и/или переключение видеорежима.
3. Непосредственный вывод в видеопамять.
4. Управление позицией курсора.

К одному компьютеру можно подключить одновременно два видеоадаптера: цветной и монохромный. Такая возможность обеспечена "разнесением" адресов экранного буфера

и адресов портов разных типов видеоадаптеров. В видеоадаптерах используются следующие адреса портов ввода/вывода:

1. VGA - от 3B0h до 3DFh
2. EGA - от 3C0h до 3CFh
3. CGA - от 3D0h до 3DFh
4. MDA - от 3B0h до 3BFh

В компьютере может быть установлено два видеоадаптера EGA. В этом случае адреса портов ввода/вывода второго адаптера лежат в области от 2C0h до 2DFh. Чтобы обеспечить работоспособность программ, рассчитанных на применение с CGA и MDA, в видеоадаптерах EGA и VGA предусмотрена возможность работы в режимах CGA и MDA. При этом говорят, что EGA и VGA "**эмулируют**" работу адаптеров CGA и MDA. Кроме того, адаптер VGA эмулирует работу EGA.

При работе в текстовом режиме видеобуфер располагается по следующим адресам:

0B800h:0000 - для цветных видеоадаптеров в цветном режиме;

0B000h:0000 - для монохромных видеоадаптеров и цветных видеоадаптеров в монохромном режиме.

Структура текстового буфера:

1 байт (четный) - код символа;
1 байт (нечетный) - атрибут символа.

Структура байта атрибута:

биты 0,1,2 - цвет символа
(0 - красный цвет)
(1 - зеленый цвет)
(2 - синий цвет)
бит 3 - яркость символа
биты 4,5,6 - цвет фона
(4 - красный цвет)
(5 - зеленый цвет)
(6 - синий цвет)
бит 7 - мигание (для EGA и VGA может быть яркость фона)

Любой другой цвет получается комбинацией перечисленных.

Текстовый режим

Программирование всех видеоадаптеров в алфавитно-цифровом режиме сводится к определению адреса видеопамяти, куда вы собираетесь записывать ваш символ, сохранению, если это необходимо, символа и атрибута, находящегося в этом месте, а затем записи вашего символа и атрибута по этому адресу. В дальнейшем, если вы сохранили символ и атрибут, вместо которого были записаны ваши значения, вы можете восстановить предыдущий вид экрана. Такой подход бывает нужен, например, при организации оконного режима.

Единственной нетривиальной задачей при программировании видеоадаптера в текстовом режиме является задача определения типа видеоадаптера, поэтому ниже она будет рассмотрена особо. Программирование видеоадаптеров предполагается подробно рассмотреть в специально посвященной данному вопросу книге.

Определение типа видеоадаптера

Функции BIOSa, обслуживающие видеоадаптер, вызываются через различные функции прерывания BIOSa INT 10h. Однако функции, которая сообщала бы программе тип видеоадаптера, среди них нет - именно поэтому такая задача и представляет известную сложность (Точнее сказать, такая функция впервые появляется для адаптера EGA). Для определения типа видеоадаптера используются следующие факты:

1. Существует совместимость видеоадаптеров "сверху вни" (т.е. адаптер VGA, помимо только своих режимов, работает и в режимах, характерных для адаптеров более низкого уровня, хотя обратное неверно).

2. Адаптеры MDA и CGA различаются по используемым адресам видеобуфера и портов ввода/вывода (порты с ударением на первом слоге - множественное число от слова порт; в отличие от портов с ударением на последнем слоге, т. е. мужских домотканых штанов).

3. Все адаптеры, за исключением HGC, основаны на микросхеме 6845 - контроллере электроннолучевой трубки, или ее развитии.

Таким образом, для определения типа видеоадаптера сначала делается попытка выполнить функцию BIOSa, определенную только для адаптера VGA; если попытка оказалась неудачной, пытаемся выполнить функцию, специфическую для адаптера EGA, а если и эта попытка не удалась, следует по адресам видеобуфера или портов ввода/вывода определить, является ли этот видеоадаптер цветным или монохромным (CGA или MDA) (см. пример 1). Как и большинство остальных примеров, пример 1 написан для трансляции под управлением транслятора MASM 5.0 и выше. Пример 1 должен быть преобразован к типу COM (для этого используется программа EXE2BIN).

Пример 1. Определение типа видеоадаптера

```
CODE_SEG        SEGMENT                 ; определение кодового сегмента
DEFINE_VIDEO    PROC FAR                ; заголовок процедуры типа far
                ASSUME      CS:CODE_SEG, DS:CODE_SEG
                                        ; назначение сегментных
                                        ; регистров (информация
                                        ; для транслятора)
                ORG         100h        ; резервируем место для PSP

BEGIN:                                  ; начало исполняемого кода программы
                XOR         AX, AX      ; очистка регистра AX
                PUSH        DS          ; сохраняем в стеке адрес возврата в систему
                PUSH        AX          ; (адрес нулевого байта PSP - DS:0000)
                JMP         START       ; переход на начало программы - обход данных
                                        ; область данных
VGA_ID          DB          0Dh, 0Ah, 9 ; это сообщение будет выведено
                DB          ' IT"S VGA ' ; на экран, если в машине установлен
                DB          0Dh, 0Ah, '$' ; адаптер VGA

EGA_ID          DB          0Dh, 0Ah, 9 ; это сообщение будет выведено
                DB          ' IT"S EGA ' ; на экран, если в машине установлен
                DB          0Dh, 0Ah, '$' ; адаптер EGA

CGA_ID          DB          0Dh, 0Ah, 9 ; это сообщение будет выведено
                DB          ' IT"S CGA ' ; на экран, если в машине установлен
                DB          0Dh, 0Ah, '$' ; адаптер CGA

MDA_ID          DB          0Dh, 0Ah, 9 ; это сообщение будет выведено
                DB          ' IT"S MDA ' ; на экран, если в машине установлен
                DB          0Dh, 0Ah, '$' ; адаптер MDA
```

```
HGC_ID          DB      0Dh, 0Ah, 9     ; это сообщение будет выведено
                DB      ' IT"S HGC '    ; на экран, если в машине установлен
                DB      0Dh, 0Ah, '$'   ; адаптер HGC

;       Начало программы определения типа видеоадаптера
START:
;       Сначала проверяем, представлен ли адаптер VGA
                MOV     AH, 1Ah         ; Попытка выполнить функцию BIOSa
                INT     10h             ; адаптера VGA (определить/установить
                                        ; состояние адаптера)
                CMP     AL, 1Ah         ; Если VGA установлен, AL=1Ah
                JNE     SEARCH_EGA      ; Если нет, проверяем установлен ли EGA
                LEA     DX, VGA_ID      ; Загружаем в DX адрес сообщения VGA_ID
                MOV     AH, 09h         ; Загружаем в AH номер функции DOSa
                INT     21h             ; Вызываем диспетчер DOSa
                RETF                    ; Возвращаем управление операционной системе

;       Проверяем, представлен ли адаптер EGA
SEARCH_EGA:
                MOV     AH, 12h         ; AH=12h и BL=10h - функция
                MOV     BL, 10h         ; BIOSa - дать информацию о наличии/статусе EGA
                INT     10h             ; вызов сервиса видеоадаптера
                CMP     BL, 10h         ; проверяем значение BL (если не изменилось ->
                                        ; EGA)
                JE      SEARCH_CGA      ; Если EGA отсутствует (флаг ZF=0), то
                                        ; переходим на проверку наличия адаптера CGA
                LEA     DX, EGA_ID      ; Загружаем в DX адрес сообщения EGA_ID
                MOV     AH, 09h         ; Загружаем в AH номер функции DOSa
                INT     21h             ; Вызываем диспетчер DOSa
                RETF                    ; Возвращаем управление операционной системе
;       Проверяем, представлен ли адаптер CGA
SEARCH_CGA:
                MOV     DX, 3D4h
                CALL    SEARCH_6845     ; Процедура SEARCH_6845 устанавливает флаг
                                        ; переноса CF, если не находит по
                JC      SEARCH_MDA      ; адресу в DX порта 6845, т.е. переход
                                        ; осуществляется в случае, если нет CGA
                LEA     DX, CGA_ID      ; Загружаем в DX адрес сообщения CGA_ID
                MOV     AH, 09          ; Загружаем в AH номер функции DOSa
                INT     21h             ; Вызываем диспетчер DOSa
                RETF                    ; Возвращаем управление операционной системе

;       Проверяем, представлен ли адаптер MDA
SEARCH_MDA:
                MOV     DX, 3B4h
                CALL    SEARCH_6845     ; Процедура SEARCH_6845 устанавливает
                                        ; флаг переноса CF, если не находит по
                JC      IT_IS_HGC       ; адресу в DX порта 6845 т.е. переход
                                        ; осуществляется в случае если нет CGA
                LEA     DX, MDA_ID      ; Загружаем в DX адрес сообщения MDA_ID
                MOV     AH, 09h         ; Загружаем в AH номер функции DOSa
```



```
                INT     21h             ; Вызываем диспетчер DOSa
                RETF                    ; Возвращаем управление операционной системе

IT_IS_HGC:
;       Нет адаптеров MDA, CGA, EGA или VGA -> представлен HGC
                LEA     DX, HGC_ID      ; Загружаем в DX адрес сообщения HGC_ID
                MOV     AH, 09h         ; Загружаем в AH номер функции DOSa
                INT     21h             ; Вызываем диспетчер DOSa
                RETF                    ; Возвращаем управление операционной системе
;***********************************************
; Подпрограмма проверки наличия CRT контроллера 6845
SEARCH_6845 PROC
                MOV     AL, 0Fh
                OUT     DX, AL          ; В DX содержится адрес порта, посылая значение
                                        ; в который, мы, тем самым выбираем регистр
                                        ; микросхемы, к которому будем обращаться (порт
                                        ; 3D4h для CGA и 3B4h для MDA - порт выбора
                                        ; регистра).
                                        ;       В этой процедуре выбирается регистр
                                        ; 0Fh - регистр положения курсора (нижнего).
                                        ; Этот регистр выбран из-за того, что он
                                        ; READ/WRITE регистр
                INC     DX              ; Увеличиваем содержимое регистра DX на 1,
                                        ; переходя тем самым к порту, через который мы
                                        ; читаем и в который пишем содержимое
                                        ; выбранного регистра CRT
                IN      AL, DX          ; Читаем содержимое регистра 0Fh в AL
                MOV     AH, AL          ; Сохраняем регистр AL в AH
                MOV     AL, 66h         ; Заносим в AL тестовое значение для проверки
                                        ; наличия микросхемы 6845
                OUT     DX, AL          ; Пишем тестовое значение в регистр CRT номер
                                        ; 0Fh
                MOV     CX, 100h        ; Организуем временную задержку,
DELAY:                                  ; необходимую для срабатывания
                LOOP    DELAY           ; микросхемы CRT 6845
                IN      AL, DX          ; Вводим содержимое регистра CRT номер 0Fh
                                        ; в регистр AL
                XCHG    AH, AL          ; Обмениваем содержимое регистров AH и AL
                OUT     DX, AL          ; Восстанавливаем содержимое регистра
                                        ; CRT номер 0Fh
                CMP     AH, 66h         ; Проверяем: произошла ли запись в микросхему
                                        ; 6845? Т.е. имеется ли по адресу в DX такая
                                        ; микросхема
                JE      QUIT            ; Если 6845 есть, следует переход на выход из
                                        ; подпрограммы
                STC                     ; При отсутствии микросхемы видеоконтроллера
                                        ; 6845 выполняется эта команда - установка флага
                                        ; переноса CF
QUIT:
                RET                     ; Возвращение управления главной программе
SEARCH_6845     ENDP                    ; Закрываем процедуру SEARCH_6845
```

```
;**************************************************
DEFINE_VIDEO    ENDP                    ; Закрываем главную процедуру
CODE_SEG        ENDS                    ; Окончание кодового сегмента
                END         BEGIN       ; Конец кода программы
```

Графический режим

Программирование видеоадаптеров в графическом режиме является задачей довольно сложной, поэтому, как и программирование в текстовом режиме, в основном будет рассматриваться в отдельной книге. Здесь будут обсуждены только некоторые вопросы, касающиеся структуры видеобуферов адаптеров EGA и CGA и доступа к ним (видеобуферам) из программы.

Структура буфера CGA

Видеобуфер CGA - это 16 килобайт оперативной (RAM) памяти, расположенной в адресном пространстве начиная с адреса B800:0000. CGA поддерживает следующие графические режимы:

1. Режим высокого разрешения (режим 6) - 640*200 точек в 2 цветах.
2. Режим среднего разрешения (режим 4) - 320*200 точек в 4 цветах.
3. Режим среднего разрешения (режим 5) - 320*200 точек в 4 цветах (используется для RGB мониторов).
4. Режим низкого разрешения - 160*100 точек в 16 цветах. Этот режим не поддерживается функциями BIOSa.

В режиме высокого разрешения один байт экранного буфера определяет восемь точек (pixel) изображения (один бит на точку, точка может быть поставлена или нет), в режиме среднего разрешения один байт определяет четыре точки (два бита на точку), а в режиме низкого разрешения - две точки изображения (по четыре бита на точку).

Особенностью структуры видеобуфера CGA, которую необходимо учитывать при использовании прямого доступа к видеобуферу этого адаптера, является разделение видеопамяти на два банка по восемь килобайт, в первом из которых, начинающемся с

адреса B800:0000, хранятся четные строчки изображения, а во втором, начинающемся с адреса B800:2000, хранятся нечетные строчки (см. рис.1).

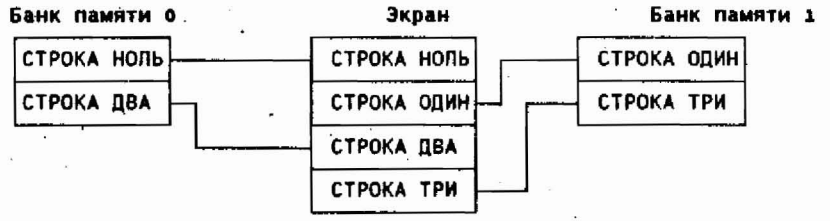


Рис 1. Структура видеобуфера CGA

При создании изображения для модификации видеобуфера, и, следовательно, изображения на экране, используются любые команды работы с оперативной памятью (команды пересылки, арифметические и логические). Одна строка изображения, вне зависимости от выбранного графического режима, всегда отображается в 80 байт видеопамяти. В самом деле, в режиме высокого разрешения адаптера CGA длина строки изображения 640 точек, а в одном байте хранится восемь точек изображения; следовательно, одна строка отображается в 640:8=80 байт (или 050h в шестнадцатиричной системе счисления). Аналогично в режиме среднего разрешения длина строки изображения 320:4=80 байт, а в режиме низкого разрешения 160:2=80 байт.

Структура буфера EGA.

Видеобуфер EGA имеет от 64K в первых версиях адаптера до 256k оперативной видеопамяти в современных вариантах адаптера. Кроме режимов, характерных для CGA, появились новые видеорежимы:

1. Режим среднего разрешения (режим 0Dh) - 320*200 точек в 16 цветах.
2. Режим высокого разрешения (режим 0Eh) - 640*200 точек в 16 цветах.
3. Режим высокого разрешения (режим 0Fh) - 640*350 точек в 2 цветах.
4. Режим высокого разрешения (режим 10h) - 640*350 точек в 16 цветах.

При использовании этих режимов (а также режимов VGA адаптера) видеобуфер расположен по адресу A000:0000. Казалось бы, от программиста требуется только определить адрес оперативной памяти и модифицировать соответствующий элементу экрана байт, однако, при программировании адаптера EGA возникают большие сложности, обусловленные строением видеопамяти. Для видеопамяти адаптеров EGA и VGA, работающих в режимах высокого разрешения, отведено адресное пространство от 0A000h:0000 до 0A000h:0FFFFh, т.е. всего 64 килобайта, а размер видеобуфера, как уже говорилось выше, достигает 256 килобайт. Если же мы подсчитаем количество оперативной памяти необходимое для хранения целого экрана при работе в режиме 10h, то получим, учитывая, что для хранения одной точки требуется половина байта, 320*350 байт или, приблизительно 110K. Кроме того адаптер EGA имеет две экранные страницы при работе в графическом режиме, что требует наличия 220K. Таким образом, отведенное для видеопамяти адресное пространство выглядит явно недостаточным для хранения изображения в режиме высокого разрешения. Становится ясным, что организация видеопамяти в адаптерах EGA и VGA (в этих режимах) является довольно сложной. В самом деле, как вместить память в 256K в отведенное для нее адресное пространство 64K? Очевидна необходимость какого то механизма для решения этой проблемы. Вот здесь и вводится механизм битовых плоскостей (bit planes). Каждая битовая плоскость занимает 64K памяти и отвечает, или за воспроизведение одного из трех цветов (красный, зеленый, синий), или за интенсивность изображения. Модифицируя информацию в видеобуфере, вы модифицируете, на деле содержимое одной или нескольких битовых плоскостей, а читая содержимое видеобуфера, опять же, читаете содержимое одной или нескольких битовых плоскостей. Что именно вы прочитаете или запишете зависит от состояния регистров видеоконтроллера, установленных режимов чтения/записи, а также, от того, какая предыдущая операция ввода/вывода выполнялась. Важно также и то, что битовые плоскости вам напрямую недоступны. Когда вы выполняете какую-либо операцию по чтению/модификации видеобуфера, вы, на деле обращаетесь к буферным регистрам-защелкам (Latches). Выполняя какую-либо команду чтения из битовых плоскостей (например с помощью

команды MOV), вы копируете содержимое соответствующих байтов битовых плоскостей в четыре регистра-защелки, а процессор получает свои данные уже из этих регистров. Выполняя команду записи (например с помощью команд OR, XOR и других, позволяющих изменять содержимое ячейки памяти), вы, на самом деле, модифицируете содержимое битовых плоскостей с помощью данных, находящихся в этих регистрах, а также, в зависимости от режима записи и состояния графических регистров контроллера, с использованием данных, находящихся в этих регистрах и переданных процессором контроллеру. Схематически передача данных между центральным процессором и адаптерами EGA и VGA представлена на рисунках 2 и 3.

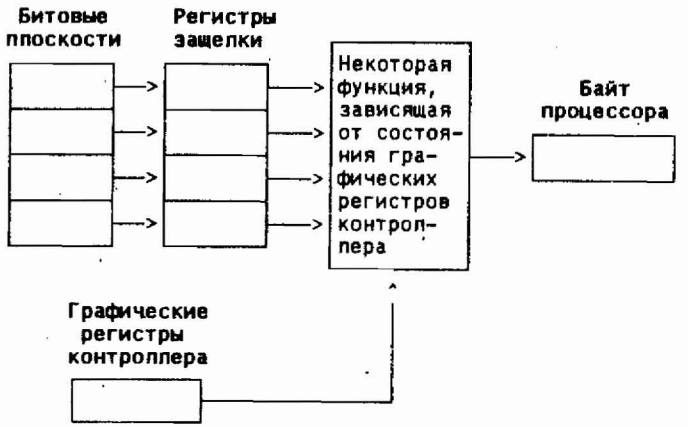


Рис 1. Чтение из видеопамяти в адаптерах EGA и VGA

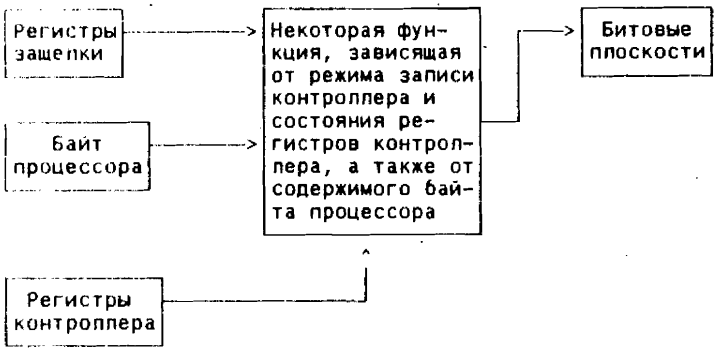


Рис 3. Запись в видеобуфер для EGA/VGA

То есть, фундаментальным отличием адаптеров, видеопамять которых организована параллельными битовыми плоскостями (такими, как EGA, VGA, Hercules InColor Card) от остальных видеоадаптеров для IBM-совместимых компьютеров является то, что невозможно напрямую обратиться к видеопамяти, подобно тому, как вы это делаете в адаптерах CGA, MCGA, MDA, HGC. Когда вы передаете данные в видеобуфер адаптера EGA и других, использующих битовые плоскости, реально данные передаются не непосредственно в видеопамять, а логическим схемам, которые в зависимости от состояния регистров контроллера распределяют данные между различными битовыми плоскостями. Важно также и то, что программист не может заполнить регистры-защелки своими данными напрямую, так как они являются программно недоступными. В этих регистрах находится результат последнего чтения из битовых плоскостей.

Регистры контроллера EGA. Режимы чтения/записи EGA

В этой главе рассматриваются графические регистры контроллера EGA, управляющие режимами работы с битовыми плоскостями (режимами чтения/записи). Все сказанное

здесь относится только к графическим режимам адаптеров EGA и VGA. В таблице перечислены регистры графических режимов и даны их значения, которые устанавливаются при начальной загрузке машины.

Таблица. ГРАФИЧЕСКИЕ РЕГИСТРЫ КОНТРОЛЛЕРОВ EGA И VGA

Регистр	Функция	Значение
0	Установка/сброс (Set/Reset)	0
1	Разрешение установки/сброса (Enable Set/Reset)	0
2	Сравнение цвета (Color Compare)	0
3	Циклический сдвиг данных (Data Rotate)	0
4	Выбор битовой плоскости (Read Map Select)	0
5	Режим (Mode)	Биты 0 - 3 равны 0
6	Добавочные функции (Miscellaneous)	Зависит от режима
7	Запрещение чтения цвета, или фильтрация (Color Don't Care)	0Fh при 16 цветах, 1Fh - 640*480 точек при 2 цветах
8	Битовая маска (Bit Mask)	0FFh

Регистр 0 (регистр установки/сброса) позволяет установить или сбросить значение байта в четырех битовых плоскостях.

Регистр 1 (разрешение установки/сброса) управляет работой регистра установки/сброса. Записывая в какой-либо бит этого регистра единицу, мы разрешаем использование соответствующего бита регистра установки/сброса.

Регистр 2 (регистр сравнения цвета) определяет цвет для режима чтения 1 (подробности описаны ниже).

Регистр 3 (регистр циклического сдвига данных/регистр выбора функций). В этом регистре независимо друг от друга работают две битовые группы: группа из трех бит 0 ... 2 определяет циклический сдвиг данных байта, передаваемого из процессора в видеопамять; сдвиг осуществляется слева направо; группа из двух бит 3 и 4 определяет логическую функцию, выполняемую при передаче в видеопамять над байтом данных процессора и содержимым буферов данных видеопамяти (если биты 3 и 4 равны 0, данные передаются в видеопамять без изменения; если бит 3 равен 1, а бит 4 равен 0, то выполняется функция

логическое "и"; если бит 4 равен 1, а бит 3 равен 0 - функция логическое "или"; если оба бита равны единице, - функция "исключающее или").

Регистр 4 (регистр выбора битовой плоскости). Используется для выбора битовой плоскости в операциях чтения из видеопамяти.

Регистр 5 (регистр режима). Предназначен для выбора режимов чтения/записи. Биты 0 и 1 устанавливают режим записи (от 0 до 3, причем режим 3 работает только в VGA); бит 2 не используется, бит 3 устанавливает режим чтения (0 или 1).

Регистр 6 (регистр добавочных функций, или регистр "разное"). Бит 0 равен 0 в при работе алфавитно-цифровом режиме контроллера, 1 - в графическом режиме; бит 1 - выбор отображаемой на экран страницы; биты 2 и 3 определяют, какие адреса видеопамяти отображаются на экран.

Ниже указано соответствие значений битов 2 и 3 адресам видеопамяти.

Бит 3	Бит 2	Адрес видеобуфера
0	0	0A0000h - 0BFFFFh
0	1	0A0000h - 0AFFFFh
1	0	0B0000h - 0B7FFFh
1	1	0B8000h - 0BFFFFh

Регистр 7 (регистр запрещения чтения цвета, или регистр фильтрации). В режиме чтения 1 запрещает передачу цвета, или набора цветов, в процессор.

Регистр 8 (регистр битовой маски). Если бит в этом регистре установлен в 1, то соответствующий пиксель байта видеобуфера модифицируется операцией записи. Этот регистр работает во всех режимах записи.

Режим чтения 0

Режим чтения 0 устанавливается по умолчанию в результате загрузки машины. В этом режиме, используя четвертый регистр (регистр выбора битовой плоскости) видеоконтроллера, вы выбираете номер битовой плоскости, которую вы будете читать. Это самый простой режим чтения для рассматриваемых видеоадаптеров. Схематически он показан на рисунке 4.

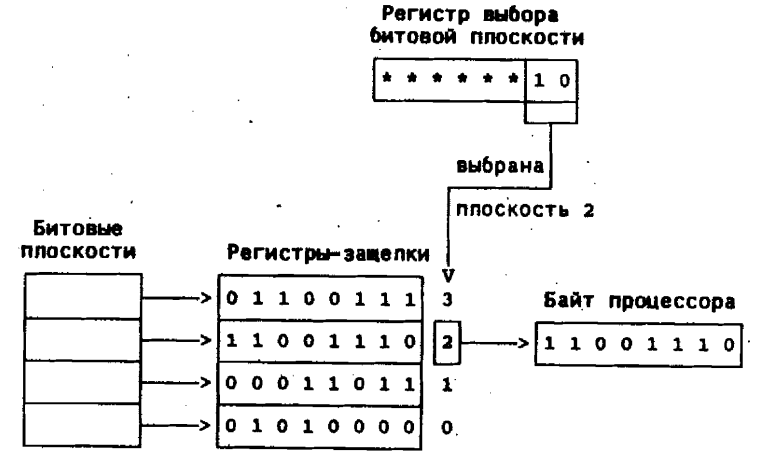


Рис 4. Режим чтения 0

Режим чтения 1

В режиме чтения 1 каждый из восьми пикселей, содержащихся в буферных регистрах, видеоадаптеров сравнивается с содержимым регистра 2, сравнения цвета. В случае совпадения содержимого регистра сравнения цвета и значения пикселя, в байте, передаваемом процессору, на соответствующем этому пикселю месте, устанавливается единица. Сравнение производится не непосредственно с пикселем, а с результатом выполнения операции логического "и" над пикселем и содержимым регистра запрещения переноса цвета. Результат сравнения передается процессору в виде одного байта. Схема получения байта в режиме чтения 1 схематически представлена на рисунке 5. Таким

образом, для чтения одного пикселя, в режиме чтения 1, требуется до 15 операций чтения. Режим чтения 1 имеет смысл использовать в тех случаях, когда вам требуется определить, совпадает ли цвет точки, с заранее заданным.

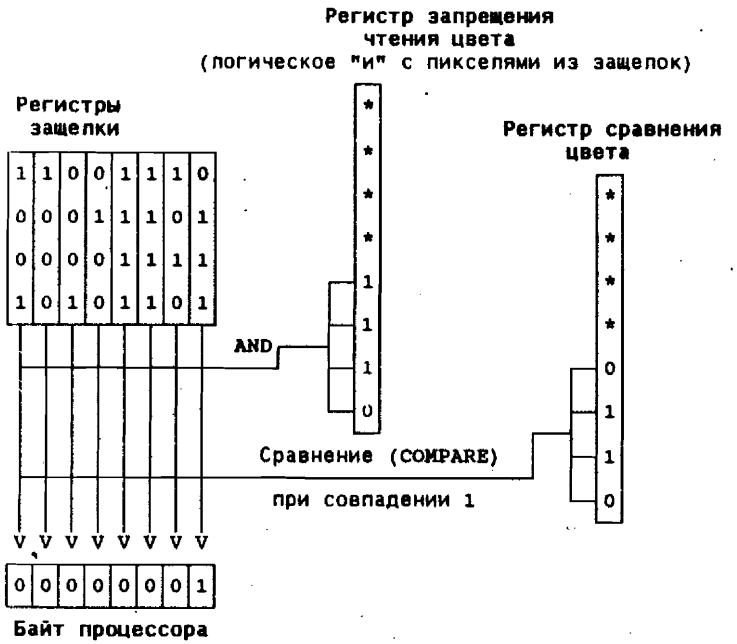


Рис 5. Режим чтения 1

Для EGA и VGA в регистрах сравнения цвета и запрещения чтения цвета работают только последние четыре бита, для адаптера класса SuperVGA, имеющего восемь битовых плоскостей и, соответственно, позволяющего работать в 256 цветах, работает весь регистр

целиком. Как ясно из рисунка регистр запрещения цвета позволяет запретить чтение какого либо цвета, то есть, фактически, отфильтровать какой то цвет.

Режим записи 0

Режим записи 0 является комбинацией байтно-ориентированной и пиксель-ориентированной модификации содержимого экранного буфера.

Регистр циклического сдвига данных/выбора функции указывает, какая именно логическая операция выполняется над данными при передаче их в буферную память (операция "ЗАМЕЩЕНИЕ", "ЛОГИЧЕСКОЕ И", "ЛОГИЧЕСКОЕ ИЛИ", "ИСКЛЮЧАЮЩЕЕ ИЛИ"), и в зависимости от состояния регистра разрешения установки/сброса, позволяет выполнить циклический сдвиг данных слева направо.

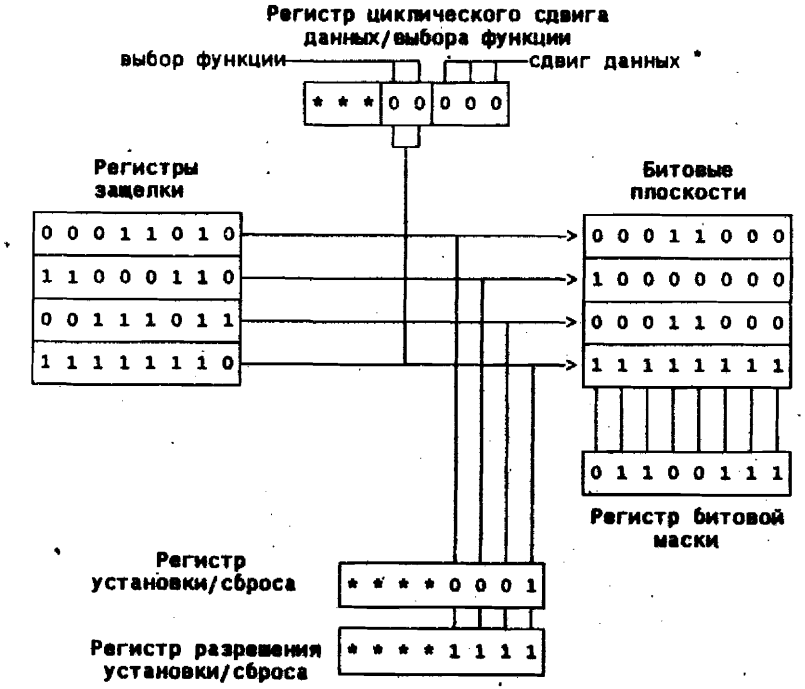
Регистр разрешения установки/сброса используется для того, чтобы выбрать, какая именно, байт или пиксель-ориентированная передача данных в видеопамять будет производиться. Если его значение равно 0Fh, то производится пиксель-ориентированная передача данных. Если же его значение равно 0h, то байт-ориентированная. Промежуточное значение этого регистра позволяет производить всякие трюки при работе с прямым обращением к видеопамяти в графических режимах .

Регистр установки/сброса. Его значение используется при замещении или модификации пикселя в пиксель-ориентированном способе передачи данных. В байт-ориентированной модификации данных режима записи 0 этот регистр не работает.

Регистр битовой маски указывает, какие именно из восьми пикселей будут модифицированы при данной операции записи. Если какой-либо бит этого регистра равен 0, то модификация этого пикселя не производится, соответствующий пиксель копируется без изменения из буферных регистров видеоадаптера.

Байт данных центрального процессора участвует в формировании изображения только в том случае, когда регистр разрешения установки/сброса равен 0, то есть только тогда, когда используется байт-ориентированная передача данных видеоадаптеру.

Схематически графический режим записи 0 представлен на рисунках 6 и 7.



* В этом режиме, если регистр разрешения установки/сброса равен ОГН, циклический сдвиг не работает

Рис 6. Режим записи 0 (пиксель-ориентированный)

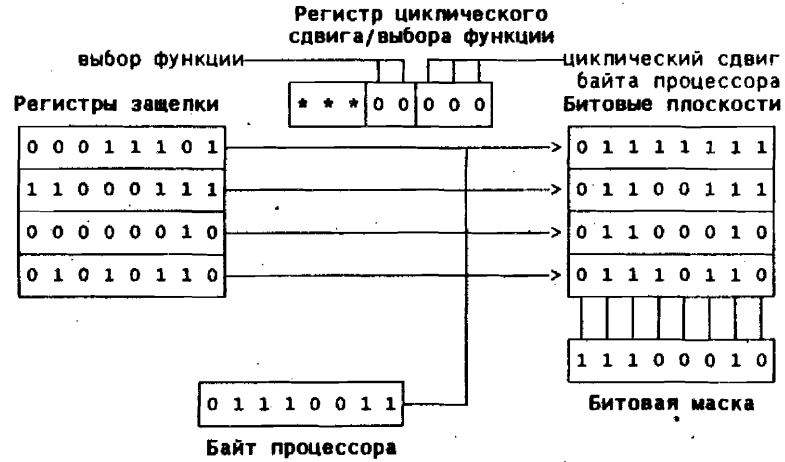


Рис 7. Режим записи 0 (байт-ориентированный)

Режим записи 1

В этом режиме записи, содержимое буферных регистров-защелок, записанное туда в результате каких-либо предыдущих операций ввода или вывода, копируется без изменений в видеопамять (См. рис.8).

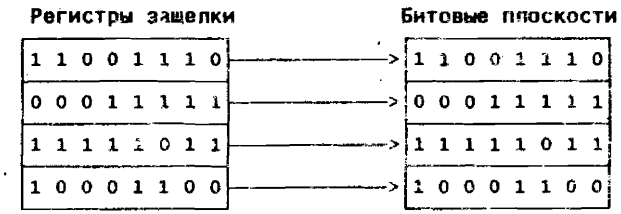
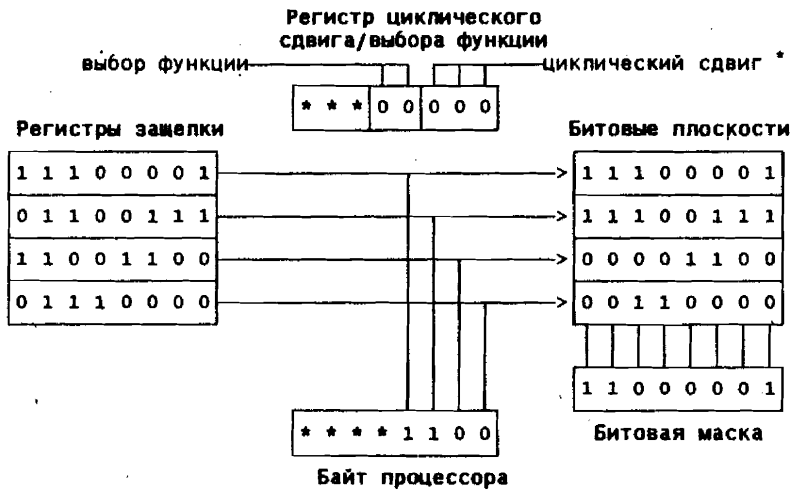


Рис 8. Режим записи 1

Режим записи 2

В этом режиме младшая половина байта процессора используется в качестве пикселя для модификации содержимого видеопамяти, причем в выработке результата участвуют регистр циклического сдвига данных/выбора функции и регистр битовой маски. Графический режим записи номер 2 аналогичен пиксель-ориентированной форме режима записи 0, в котором вместо регистра установки/сброса участвует байт процессора (См. рис.9).

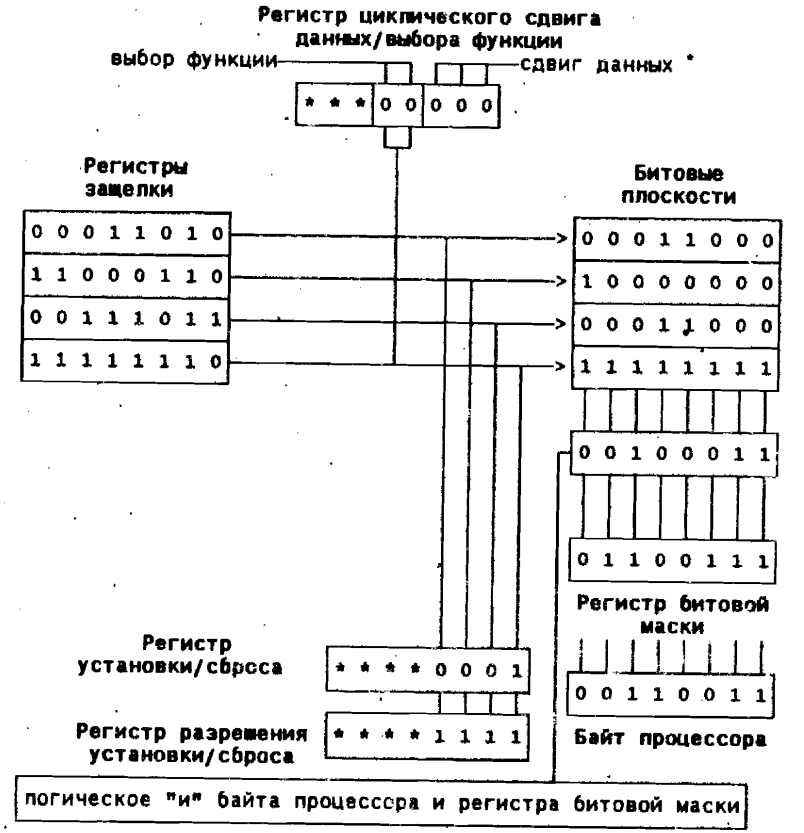


* В этом режиме циклический сдвиг не работает

Рис 9. Режим записи 2

Режим записи 3 (только VGA)

Графический режим записи 3 поддерживается только адаптерами VGA. Этот режим отличается от пиксель-ориентированного режима записи 0 тем, что в качестве битовой маски используется результат логического "и" байта процессора, сдвинутого на указанное в регистре циклического сдвига число байт, и регистра битовой маски (См. рис.10).



* Действует на байт процессора

Рис 10. Режим записи 3 (только VGA)

Чтение/модификация видеобуфера

Для того, чтобы прочитать или модифицировать содержимое видеобуфера, надо сначала выбрать режим чтения/записи, а затем, если это необходимо, запрограммировать графические регистры видеоконтроллера. Сначала выбираем регистр, который собираемся модифицировать. Регистр выбирается записью его номера в порт 03CEh. После этого значение, которое мы собираемся записать в регистр, заносится в порт 03CFh. В качестве примера показан фрагмент программы, в котором выбирается режим чтения 1 и режим записи 2.

Пример 2. Выбор режима чтения/записи для EGA

```
MOV   DX, 03CEh     ; заносим в DX адрес порта выбора графических регистров
MOV   AL, 05h       ; в AL заносим номер регистра видеорежима
OUT   DX, AL        ; в порт 03CEh посылаем из AL номер регистра
MOV   DX, 03CFh     ; в DX адрес порта данных графических регистров
MOV   AL, 1010B     ; бит 3 = 1 - режим чтения 1 биты 0-1 = 10B - режим записи 2
OUT   DX, AL        ; в порт 03CFh посылаем из AL содержимое регистра выбора видео
                    ; режима
```

Используя операции ассемблера со словами, можно сделать то же самое за три команды (пример 2а).

Пример 2а

```
MOV   DX, 03CEh     ; заносим в DX адрес порта
MOV   AX, 0A05h     ; заносим в AL номер регистра, а в Ah выбранный режим чтения и записи
OUT   DX, AX        ; посылаем в порты 3CEh и 3CFh содержимое регистра AX
```

Следующий пример - чтение байта из выбранной битовой плоскости. Используется режим чтения 0. Мы будем читать байт из первой битовой плоскости. Несмотря на то, что в этом примере мы только читаем, мы должны установить и режим записи, так как он устанавливается двумя битами в том же графическом регистре адаптера, что и режим чтения (регистр 5 - регистр режима).

Пример 3. Чтение байта из битовой плоскости, для адаптеров EGA/VGA в режиме чтения 0

```
MOV   DX, 3CEh              ; заносим в DX адрес порта
MOV   AX, 0205h             ; посылаем в AL:
                            ; номер регистра - 05
                            ; (регистр выбора режима);
                            ; посылаем в Ah 02:
                            ; режим чтения - 0
                            ; (бит 3 равен 0);
                            ; режим записи - 2
                            ; (биты 0-1 равны 10B)
OUT   DX, AX                ; устанавливаем режимы чтения/записи
MOV   AX, 0A000h            ; пересылаем через AX в ES
MOV   ES, AX                ; начальный адрес видеобуфера
MOV   AL, 4                 ; выбираем регистр 4 (регистр
OUT   DX, AL                ; выбора битовой плоскости)
INC   DX                    ; переходим к порту 3CFh
MOV   AL, 1                 ; выбираем битовую
OUT   DX, AL                ; плоскость 1
MOV   AL, BYTE PTR ES:[0FE5h] ; читаем из первой битовой плоскости в AL байт
```

Следующий пример посвящен записи одного пикселя с использованием режима записи 2. Рассматривается режим работы видеоадаптера EGA 640*350 точек в 16 цветах. Из сказанного выше следует, что один байт битовой плоскости определяет цветовую составляющую восьми пикселей. Таким образом, одна цветовая составляющая одной строки занимает 80 байт одной из битовых плоскостей. Этот факт мы учтем в примере. Как видно из описания режимов записи/чтения видеобуфера, во всех режимах записи на экран участвуют буферные регистры - "защелки". Однако, запись в эти регистры происходит только при чтении из видеобуфера. Это также учтено в примере.

Пример 4. Вывод точки на экран

```
;       Входные данные подпрограммы: в регистре DX номер строки, в регистре CX номер колонки,
;       в регистре AL цвет точки.
;       Первая часть программы вычисляет адрес в экранном буфере, куда мы заносим свое значения
;       пикселя.
;       Для вычисления адреса используются координаты точки из регистров DX и CX.

Set_Pixel       PROC    NEAR            ; заголовок подпрограммы типа NEAR
;       сохраняем регистры, используемые в подпрограмме
                PUSH    AX
                PUSH    BX
                PUSH    CX
```

```
        PUSH    DX
        PUSH    ES

        MOV     BX, 0A000h      ; пересылаем через BX в ES
        MOV     ES, BX          ; начальный адрес видеобуфера

        MOV     BL, AL          ; сохраняем цвет точки в BL
        MOV     AX, 80          ; Заносим в AX длину экранной строки в байтах
        PUSH    DX              ; сохраняем в стеке DX, подготавливая операцию умножения
        MUL     CX              ; умножаем AX на количество строк
        POP     DX              ; восстанавливаем DX
        PUSH    DX              ; сохраняем в стеке DX
```

с помощью трех операций сдвига делим DX на восемь, вычисляя, тем самым, смещение модифицируемого байта относительно начала строки

```
        SHR     DX, 1
        SHR     DX, 1
        SHR     DX, 1

        ADD     AX, DX          ; вычисляем смещение модифицируемого байта относительно
                                ; начала (реально - адреса 4 параллельных байтов, по одному из
                                ; каждой битовой плоскости)
        XCHG    AX, BX          ; обмен содержимого AX и BX
        POP     DX              ; восстанавливаем DX
        AND     DX, 0111B       ; вычисляем положение пикселя в байте
        POP     AX              ; восстанавливаем в AL цвет точки
        MOV     AH, AL          ; сохраняем в AH цвет точки
        PUSH    DX              ; сохраняем в стеке DX
        MOV     DX, 03CEh       ; адрес порта выбора графического регистра занесли в
                                ; регистр DX
        MOV     AL, 5           ; заносим в AL номер регистра выбора режима чтения/записи
        OUT     DX, AL          ; послали в порт выбора графического регистра чтения/записи
                                ; 5 - номер регистра режима
        INC     DX              ; адрес порта передачи данных графических регистров в DX
        MOV     AL, 1010B       ; в AL выбранный режим записи/чтения
        OUT     DX, AL          ; выбрали режим чтения/записи
        POP     DX              ; восстанавливаем DX (в регистре DL число, указывающее
                                ; положение пикселя в байте)
```

; сейчас в регистре BX смещение модифицируемого байта от начала экранного буфера, в
; регистре AH цвет пикселя, в регистре DL положение пикселя в байте

```
        MOV     CL, DL          ; счетчик сдвигов для операции SHR
        MOV     DL, 80h         ; в регистр DL посылаем битовую маску для работы с регистром
                                ; битовой маски
        SHR     DL, CL          ; устанавливаем 1 в той позиции, куда мы будем записывать
                                ; пиксель
        MOV     CL, DL          ; сохраняем регистр DL
        MOV     DX, 03CEh       ; адрес порта выбора графического регистра занесли в регистр
                                ; DX
        MOV     AL, 3           ; заносим в AL номер регистра циклического сдвига/выбора
                                ; функции
```



```
        OUT     DX, AL          ; послали в порт выбора регистра 3 - номер регистра
                                ; циклического сдвига/выбора функции
        INC     DX              ; адрес порта передачи данных графических регистров в DX
        MOV     AL, 0           ; в AL 0 - нет сдвига, замещение пикселя
        OUT     DX, AL          ; записали в регистр циклического сдвига 0
        MOV     DX, 03CEh       ; адрес порта выбора графического регистра занесли в
                                ; регистр DX
        MOV     AL, 8           ; заносим в AL номер регистра битовой маски
        OUT     DX, AL          ; послали в порт выбора регистра 8 - номер регистра битовой
                                ; маски
        INC     DX              ; адрес порта передачи данных графических регистров в DX
        MOV     AL, DL          ; в AL битовая маска для выбора модифицируемого пикселя
        OUT     DX, AL          ; пересылаем в регистр битовой маски битовую маску
```

; мы "подготовили почву" для записи пикселя в видеобуфер: сейчас в регистрах BX смещение,
; в ES сегмент видеобуфера, в AH значение пикселя. Теперь, чтобы гарантировать запись
; правильного значения мы должны скопировать содержимое четырех параллельных байтов
; битовых плоскостей, которые мы собираемся модифицировать, в буферные регистры
; "защелки" (Latches)

```
        MOV     AL, ES:[ BX ]
        MOV     ES:[ BX ], AH   ; записали пиксель в видеобуфер

;       восстанавливаем регистры из стека
        POP     ES
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        RET                     ; возвращение в основную программу
Set_Pixel   ENDP                ; конец кода подпрограммы
```

Работа с дисками

Первым адаптером жестких дисков является адаптер MFM. Максимальная скорость передачи данных, для этого типа адаптеров, - 5 Мбит/сек, каждый трек разбит на 17 секторов по 512 байт в секторе. Однако, как емкость, так и скоростные качества этого адаптера считаются сейчас неудовлетворительными, в связи с чем в настоящее время разработано большое количество адаптеров с улучшенными характеристиками. Размер сектора для всех адаптеров остается 512 байт. Адаптер типа RLL имеет скорость передачи данных 7.5 Мбит/сек, а количество секторов на дорожке может быть от 26 до 33. Адаптеры IDE, SCSI, ESDI, кроме того имеют интеллектуальный интерфейс с CPU (Central

Processor Unit - центральное процессорное устройство, центральный процессор), что позволяет существенно повысить скорость обмена данными. В этих адаптерах количество секторов на одной дорожке доходит до 60. Используя адаптер типа RLL, можно отформатировать диск предназначенный для работы с MFM адаптером, увеличив тем самым его емкость примерно в 1.5 раза, однако, в связи с более жесткими требованиями предъявляемыми к RLL устройствам надежность хранения информации, в этом случае не гарантируется. Адаптер ESDI обычно применяется (и разработан) в старших моделях IBM PS2. Адаптер SCSI, строго говоря, не является чисто дисковым адаптером, а позволяет подключать до восьми разнообразных устройств, в том числе накопители на магнитных лентах. Сервис BIOSa для дисков осуществляется при помощи функций прерывания INT 13h.

Как правило, дисковым сервисом BIOSa программисты пользуются в случаях какого-то нестандартного использования дисковых устройств, таких, например, как защита данных от несанкционированного копирования, доступ к первому сектору нулевой дорожки на нулевой стороне жесткого диска и других. Для защиты данных от несанкционированного доступа обычно используют, либо необычное форматирование какой-либо области на диске, либо жесткую привязка программы к её положению на диске, либо объявление какого нибудь элемента дискового пространства плохим (bad sector, bad claster, bad track). В первом секторе на нулевой дорожке и нулевой стороне жест-кого диска находится так называемая "программа предзагрузки", в которой содержится, в частности, таблица разделов жесткого диска (Disk Partitions Table). Эта таблица управляет разделением физического устройства - жесткого диска - на логические устройства, используемые операционной системой (DOS). Каждое логическое устройство (логический диск) по строению и использованию операционной системой эквивалентно дискете. Таблица разделов жесткого диска расположена со смещением 1BEh от начала сектора и состоит из четырех 16-байтовых элементов, описывающих дисковые разделы (Disk Partitions), и двухбайтового признака конца таблицы. Ниже приведена структура таблицы разделов, расположение её на дорожке и структура одного описателя раздела.

Таблица разделов жесткого диска.

смещение	размер	содержание
0h	1BEh	код программы, загружающей в оперативную память и запускающей загрузочный сектор активного раздела
1BEh	10h	описатель раздела
1CEh	10h	описатель раздела
1DEh	10h	описатель раздела
1EEh	10h	описатель раздела
1FEh	02h	0AA55h - признак конца таблицы

Структура описателя раздела

смещение	размер	содержание
0	1	флаг загрузки (если он равен 1, то загружается в память и стартует загрузочный сектор этого раздела)
1	1	номер головки начала раздела
2	2	начальный сектор и цилиндр, причем биты с нулевого по пятый включительно это номер сектора, а биты с шестого по пятнадцатый - номер цилиндра
4	1	код системы: 0-неизвестная, 1-DOS с 12-и битовой таблицей размещения файлов (FAT), 4-DOS с 16-и битовой таблицей размещения файлов, 5-элемент расширенной таблицы разделов (MS-DOS версий 4 и 5. Служит для преодоления 32 мегабайтного барьера емкости жесткого диска)
5	1	номер головки конца раздела
6	2	конечный сектор и цилиндр
8	4	номер стартового относительного сектора (т.е. сектора DOS)
0Ch	4	размер раздела в секторах
10h		начало следующего определителя раздела, или 0AA55h

Программа, читающая первый сектор первой дорожки и выводящая на экран таблицу разделов жесткого диска, приведена в примере 5.

Пример 5. Чтение таблицы разделов жесткого диска

```
B_P     EQU   BYTE PTR   ; создали новую мнемонику для
                         ; обозначения указателя байта
W_P     EQU   WORD PTR   ; создали новую мнемонику для
                         ; обозначения указателя слова
P_1     EQU   1BEh       ; смещение от начала сектора
                         ; до описателя раздела 1
P_2     EQU   1CEh       ; смещение от начала сектора
                         ; до описателя раздела 2
P_3     EQU   1DEh       ; смещение от начала сектора
                         ; до описателя раздела 3
```

```
P_4         EQU   1EEh          ; смещение от начала сектора
                                 ; до описателя раздела 4

        макроопределение расшифровки описателя раздела
TRANSLATE   MACRO       P,HEAD,SEC,CYL      ; параметры макроопределения:
                                            ; P - смещение обрабатываемых
                                            ; данных от начала сектора
                                            ; HEAD - адрес для информации
                                            ; о номере головки
                                            ; SEC - адрес для информации
                                            ; о номере сектора
                                            ; CYL - адрес для информации
                                            ; о номере цилиндра
            MOV   AL, B_P [ SI+P+1 ]        ; записали в AL номер головки раздела
            XLAT                            ; перекодируем
            MOV   HEAD, AL                  ; номер головки послали из AL в выводимую строку
            MOV   AX, W_P [ SI+P+2 ]        ; (номер цилиндра раздела + номер сектора раздела)
                                            ; посылаем в регистр AX
            MOV   DX, AX                    ; сохранили AX в DX
            AND   AX, 0111111B              ; выделили информацию о номере сектора
            MOV   AH, AL                    ; сохранили AL в AH
            AND   AL, 0Fh                   ; выделили младшую цифру
            XLAT                            ; перекодируем
            XCHG  AL, AH                    ; обменяли содержимое регистров AL и AH
            MOV   CL, 4                     ; подготавливаем счетчик для сдвига
            SHR   AL, CL                    ; сдвигаем AL на 4 разряда вправо (делим на 16)
            XLAT                            ; перекодировали AL в символьную форму
            MOV   SEC, AL                   ; номер сектора послали
            MOV   SEC + 1, AH               ; в выводимую строку
            AND   DX, 0FFC0h                ; выделили информацию о номере цилиндра
            MOV   CL, 6                     ; подготавливаем счетчик для сдвига
            SHR   DL, CL                    ; получаем старшую цифру номера цилиндра
            MOV   AL, DL                    ; содержимое DL посылаем в AL для подготовки
                                            ; команды XLAT
            XLAT                            ; перекодировали AL
            MOV   CYL, AL                   ; старшую цифру номера цилиндра послали в выводимую
                                            ; строку
            MOV   AL, DH                    ; помещаем в AL две младшие цифры номера цилиндра

            MOV   CL, 4                     ; подготавливаем счетчик для сдвига
            SHR   AL, CL                    ; выделяем вторую цифру номера цилиндра
            XLAT                            ; перекодировали AL
            MOV   CYL + 1, AL               ; вторую цифру номера цилиндра послали в
                                            ; выводимую строку
            MOV   AL, DH                    ; помещаем в AL две младшие цифры номера цилиндра
            AND   AL, 0Fh                   ; выделили младшую цифру номера цилиндра
            XLAT                            ; перекодировали AL
            MOV   CYL + 2, AL               ; последнюю цифру номера цилиндра послали в
                                            ; выводимую строку
            ENDM
```



```
;       ***********************************************************
;       начало программы
CODE        SEGMENT                     ; определение кодового сегмента
PARTITION   PROC    FAR                 ; заголовок процедуры типа far
            ASSUME  CS:CODE             ; назначение сегментных регистров (информация для
                                        ; транслятора)
            ORG     100h                ; резервируем место для PSP
BEGIN:                                  ; начало кода программы
            XOR     AX, AX              ; очистка регистра AX
            PUSH    DS                  ; сохраняем в стеке адрес возврата
            PUSH    AX                  ; (адрес 0 байта PSP - DS:0000)
            JMP     START               ; переход на начало программы
;       область данных
REPORT      DB      0Dh, 0Ah, 9, 9
            DB      'Таблица разделов жесткого диска'
            DB      0Dh, 0Ah
REPORT1_S   DB      'начало раздела 1'
            DB      9, 'головка: '
HEAD1_S     DB      ?,'h'
            DB      9, 'цилиндр: '
CYL1_S      DB      ?,?,?,'h'
            DB      9, 'сектор: '
SEC1_S      DB      ?,? 'h'

            DB      0Dh, 0Ah
REPORT1_E   DB      'конец раздела 1'
            DB      9, 'головка: '
HEAD1_E     DB      ?,'h'
            DB      9, 'цилиндр: '
CYL1_E      DB      ?,?,?,'h'
            DB      9, 'сектор: '
SEC1_E      DB      ?,?,'h'
            DB      0Dh, 0Ah
            DB      0Dh, 0Ah

REPORT2_S   DB      'начало раздела 2'
            DB      9, 'головка: '
HEAD2_S     DB      ?,'h'
            DB      9, 'цилиндр: '
CYL2_S      DB      ?,?,?,'h'
            DB      9, 'сектор: '
SEC2_S      DB      ?,?,'h'
            DB      0Dh, 0Ah
REPORT2_E   DB      'конец раздела 2'
            DB      9, 'головка: '
HEAD2_E     DB      ?,'h'
            DB      9, 'цилиндр: '
CYL2_E      DB      ?,?,?,'h'
            DB      9, 'сектор: '
SEC2_E      DB      ?,?,'h'
            DB      0Dh, 0Ah
```

```
                DB      0Dh, 0Ah
REPORT3_S       DB      'начало раздела 3'
                DB      9, 'головка: '
HEAD3_S         DB      ?,'h'
                DB      9, 'цилиндр: '
CYL3_S          DB      ?,?,?,'h'
                DB      9, 'сектор: '
SEC3_S          DB      ?,?,'h'

                DB      0Dh, 0Ah
REPORT3_E       DB      'конец раздела 3'
                DB      9, 'головка: '
HEAD3_E         DB      ?,'h'
                DB      9, 'цилиндр: '
CYL3_E          DB      ?,?,?,'h'
                DB      9, 'сектор: '
SEC3_E          DB      ?,?,'h'
                DB      0Dh, 0Ah
                DB      0Dh, 0Ah

REPORT4_S       DB      'начало раздела 4'
                DB      9, 'головка: '
HEAD4_S         DB      ?,'h'
                DB      9, 'цилиндр: '
CYL4_S          DB      ?,?,?,'h'
                DB      9, 'сектор: '
SEC4_S          DB      ?,?,'h'

                DB      0Dh, 0Ah
REPORT4_E       DB      'конец раздела 4'
                DB      9, 'головка: '
HEAD4_E         DB      ?,'h'
                DB      9, 'цилиндр: '
CYL4_E          DB      ?,?,?,'h'
                DB      9, 'сектор: '
SEC4_E          DB      ?,?,'h'
                DB      0Dh, 0Ah, '$'
;       конец информационной строки

READ_BUFF       DB      200h DUP(?)             ; буфер для чтения первого сектора жесткого
                                                ; диска

TABLE           DB      '0123456789ABCDEF'      ; таблица для перекодировки из числового
                                                ; шестнадцатиричного представления в символьное

START:
;       с помощью функции BIOSa читаем первый сектор жесткого диска
                MOV     DL, 80h                 ; 80h - выбираем жесткий диск 0
                MOV     DH, 0                   ; номер головки - 0
                MOV     CH, 0                   ; номер цилиндра - 0
```



```
                MOV     CL, 1                   ; номер сектора - 0
                MOV     AL, 1                   ; количество секторов читаемое за одну операцию
                                                ; (не больше, чем есть на одной дорожке)
                MOV     AH, 2                   ; выбор функции BIOSa - чтение сектора
                LEA     BX,READ_BUFF            ; заносим адрес буфера
                                                ; предназначенного для прочитанного сектора
                                                ; (должен быть по адресу в регистровой паре
                                                ; ES:BX)
                INT     13h                     ; вызываем сервис BIOSa

                MOV     SI, BX                  ; скопировали адрес начала буфера в SI
                LEA     BX, TABLE               ; посылаем в BX адрес таблицы перекодировки

;       вызываем макрокоманду TRANSLATE для расшифровки содержимого таблицы разделов
;       жесткого диска и заполнения выводимой на экран строки
                TRANSLATE P_1,HEAD1_S,SEC1_S,CYL1_S
                .SALL                           ; директива ассемблера подавляющая вывод в
                                                ; листинг генерируемого текста макроопределения
                                                ; (от английского Supress All - подавить все)
                TRANSLATE P_1+4,HEAD1_E,SEC1_E,CYL1_E
                TRANSLATE P_2,HEAD2_S,SEC2_S,CYL2_S
                TRANSLATE P_2+4,HEAD2_E,SEC2_E,CYL2_E
                TRANSLATE P_3,HEAD3_S,SEC3_S,CYL3_S
                TRANSLATE P_3+4,HEAD3_E,SEC3_E,CYL3_E
                TRANSLATE P_4,HEAD4_S,SEC4_S,CYL4_S
                TRANSLATE P_4+4,HEAD4_E,SEC4_E,CYL4_E

;       вывод на экран содержимого таблицы разделов
                MOV     AH, 9                   ; выбираем функцию DOSa 9
                LEA     DX, REPORT              ; заносим в DX адрес сообщения
                INT     21h                     ; вызов диспетчера DOSa

;***************************************************
                RETF                            ; возвращаем управление в DOS
PARTITION       ENDP                            ; Закрываем главную процедуру
CODE            ENDS                            ; Окончание кодового сегмента
                END     BEGIN                   ; Конец кода программы
```

Программа, нестандартным образом форматирующая дорожку на дискете, приведена в примере 6. В качестве нестандартного параметра выбрано количество секторов на дорожке. Программа форматирует девятую дорожку дискеты на восемь секторов. Это количество секторов хоть и входит в число стандартных для IBM-совместимых компьютеров, однако в настоящее время используется крайне редко. Пример должен быть приведен к типу COM.

Пример 6. Нестандартное форматирование дорожки дискеты

```
FORM    SEGMENT                 ; определение кодового сегмента
FORMAT  PROC    FAR             ; заголовок процедуры типа FAR
        ASSUME  CS:FORM, ES:FORM ; назначение сегментных регистров
                                ; (информация для транслятора)
        ORG     100h            ; резервируем место для PSP
BEGIN:
        XOR     AX, AX          ; запоминаем адрес
        PUSH    DS              ; PSP для возврата
        PUSH    AX              ; в DOS после окончания программы
        JMP     START           ; обходим область данных

BUFFER  DB      512 DUP(0)      ; буфер описателей форматируемой дорожки
ERR     DB      0               ; счетчик ошибок при форматировании
TEXT    DB      'плохая дискета','$' ; сообщение при ошибке форматирования


START:
        MOV     AH,0            ; выполнить функцию "сброс"
        INT     13h             ; для дисковода

        PUSH    ES              ; сохранить адрес сегмента
        MOV     AX,0000         ; присвоить ES:BX
        MOV     ES,AX           ; адрес таблицы
        MOV     BX,0078h        ; параметров дискеты
        MOV     BYTE PTR ES:[BX+4],8 ; заносим в поле "номер последнего сектора" 8
        POP     ES              ; восстанавливаем ES из-за опасности работы в
                                ; нулевом сегменте
        LEA     BX,BUFFER       ; загрузка адреса буфера описателей
                                ; форматируемой дорожки
        MOV     AH,1            ; номер описываемого сектора
        MOV     CX,8            ; счетчик цикла

;       начало цикла формирования таблицы описателей секторов
LOOP_BEGIN:                     ; (метка начала цикла)
        MOV     BYTE PTR [BX],9 ; девятая дорожка
        MOV     BYTE PTR [BX+1],0 ; нулевая сторона
        MOV     BYTE PTR [BX+2],AH ; номер сектора
        MOV     BYTE PTR [BX+3],2 ; размер сектора (2 соответствует 512B)
        ADD     BX,4            ; переход к следующему сектору
        INC     AH              ; номер следующего сектора
        LOOP    LOOP_BEGIN      ; переход вверх по программе
;       конец цикла

        LEA     BX,BUFFER       ; загрузка адреса буфера
        MOV     DL,0            ; нулевой дисковод
        MOV     DH,0            ; нулевая сторона
        MOV     CH,9            ; девятая дорожка
        MOV     AL,8            ; восемь форматируемых секторов
```



```
INTER:
        MOV     AH,5            ; вызов функции форматирования
        INT     13h             ; дорожки дискеты
        JNC     DISK_TABLE      ; в случае удачного форматирования переход к
                                ; таблице параметров дискеты
        MOV     AH,0            ; выполнить функцию
        INT     13h             ; "сброс" для дисковода
        ADD     ERR,1           ; увеличить счетчик ошибок
        CMP     ERR,3           ; если три ошибки, то дискету нельзя использовать
        JE      ERR_PRINT       ; переход к печати сообщения
        JMP     INTER           ; повторная попытка форматирования

ERR_PRINT:                      ; действия при ошибке форматирования
        LEA     DX,TEXT         ; загрузка адреса сообщения
        MOV     AH,09h          ; вызов функции DOS
        INT     21h             ; вывода на экран

DISK_TABLE:
        MOV     AX,0000         ; вернуть таблицу
        MOV     ES,AX           ; параметров дискеты
        MOV     BX,0078h        ; в исходное
        MOV     BYTE PTR ES:[BX+4],9 ; состояние
        RETF                    ; возврат в DOS
FORMAT  ENDP                    ; конец главной процедуры
FORM    ENDS                    ; конец кодового сегмента
        END     BEGIN           ; конец программы
```

Эта программа может быть использована в качестве несложного средства защиты дискеты от копирования. Для этого она должна быть дополнена программой (подпрограммой) проверяющей количество секторов на используемой дискете. Так, для данного примера, в случае, если на дискете возможно прочитать девятый сектор на девятой дорожке, можно сделать вывод, что дискета не является оригинальной. В примере 7 производится именно такая проверка. Пример 7 также должен быть приведен к типу COM.

Пример 7. Простая защита от копирования. Совместно с примером 6

```
TFORM   SEGMENT                 ; определение кодового сегмента
TFORMAT PROC    FAR             ; заголовок процедуры
        ASSUME  CS:TFORM, ES:TFORM ; назначение сегментных регистров
                                ; (информация для транслятора)
        ORG     100h            ; резервируем место для PSP. Начало исполняемого
                                ; кода программы
BEGIN:
        XOR     AX, AX          ; запоминаем адрес
        PUSH    DS              ; PSP для возврата
```

```
                PUSH    AX              ; в DOS после
                JMP     START           ; окончания программы

BUFFER          DB      512 DUP(0)      ; буфер для считывания сектора дискеты
ERR             DB      0               ; счетчик ошибок
TEXT            DB      'несанкционированное копирование','$'
                                        ; текст, выводимый при несанкционированном
                                        ; копировании
START:
                CALL    MAIN
                JNC     QUIT            ; если флаг переноса сброшен, то продолжение
                                        ; работы скопированной программы

ERR_PRINT:
                LEA     DX,TEXT         ; загружаем адрес выводимого на экран текста
                MOV     AH,09h          ; вызов функции DOSa для
                INT     21h             ; вывода на экран

QUIT:
                RET                     ; возврат в DOS
TFORMAT         ENDP

MAIN            PROC    NEAR            ; процедура проверки отсутствия девятого сектора
                                        ; на девятой дорожке

STP:
                MOV     AH,0            ; функция BIOSa установки
                INT     13h             ; контроллера дискеты в исходное состояние
                LEA     BX,BUFFER       ; загрузка адреса буфера для считывания сектора

                MOV     DL,0            ; номер дисковода
                MOV     DH,0            ; номер стороны дискеты
                MOV     CH,9            ; номер дорожки
                MOV     CL,9            ; номер сектора
                MOV     AL,1            ; число считываемых секторов

INTER:
                MOV     AH,2            ; функция BIOSa: прочи-
                INT     13h             ; тать сектор дискеты
                JC      TEST_SEC        ; если выставлен флаг ошибки, то проверяем
                                        ; какая именно ошибка
                JMP     ERRO            ; иначе - это неразрешенная копия

TEST_SEC:
                CMP     AH,4            ; на дискете нет заданного сектора ?
                JE      ST_RET          ; нет сектора
                ADD     ERR,1           ; увеличиваем счетчик ошибок
                CMP     ERR,3           ; ошибок больше трех ?
                JBE     STP             ; да -повторный запуск процедуры

ERRO:
```



```
                STC                     ; нет -установка флага переноса
                JMP     QUI             ; выход из процедуры
ST_RET:
                CLC                     ; сброс флага переноса

QUI:
                RET                     ; возврат в основную программу
MAIN            ENDP                    ; закрываем процедуру
TFORM           ENDS                    ; окончание кодового сегмента
                END     BEGIN           ; конец кода программы
```

Последовательный порт (RS-232)

Последовательный порт используется для подключения медленных внешних устройств, таких, как графопостроители, принтеры (все это можно подключать и через параллельный порт), модемы, дигитайзеры и так далее. Последовательный порт обычно используется для подключения устройства ввода "мышь". Часто последовательный порт используется также и для связи и передачи данных между машинами. Сервис BIOSa для RS-232 осуществляется использованием функций прерывания INT 14h. Однако и для последовательного порта часто используются методы непосредственного программирования контроллера, минуя обращение к функциям BIOSa, что позволяет существенно повысить скорость передачи данных через RS-232. Максимальная скорость, достижимая при использовании BIOSa - 9600 бод (бит/с). Чтобы изменить скорость обмена данными, минуя BIOS, необходимо обратиться непосредственно к регистрам микросхемы контроллера I8250 (советского аналога не имеется), через порты ввода/вывода компьютера. Для портов управления последовательным адаптером установлены следующие адреса в адресном пространстве портов ввода/вывода компьютера:

3F8h - 3FEh для COM1;
2F8h - 2FEh для COM2.

Управляют скоростью передачи два регистра - делитель частоты передачи. Обращение к младшему байту делителя происходит через порт 3F8h, а к старшему байту, через порт 3F9h. Предварительно необходимо в регистр управления линией записать 1 в старший разряд. Обращение к этому регистру производится через порт 3FBh. Программируя

эту микросхему, BIOS устанавливает тем самым между содержимым делителя частоты и скоростью передачи следующее соответствие:

Делитель		Скорость обмена
десятичный	шестнадцатиричный	бит/сек
1047	417	110
768	300	150
384	180	300
192	0C0	600
96	60	1200
48	30	2400
24	18	4800
12	0Ch	9600

Любое нестандартное значение, выходящее за пределы этой таблицы, программист должен устанавливать самостоятельно, непосредственно обращаясь к микросхеме контроллера. В качестве примера приведен фрагмент программы, устанавливающий скорость обмена 19200 бод (бит/сек):

```
MOV     DX, 3FBh        ; записали в DX адрес порта регистра
                        ; управления линией
MOV     AL, 80h         ; установили в регистре AL старший бит в единицу
OUT     DX, AL          ; установили старший бит в единицу в регистре
                        ; управления линией
MOV     DX, 3F8h        ; записали в DX адрес порта делителя частоты
                        ; обмена
MOV     AX, 06          ; записали в AX коэффициент деления
OUT     DX, AX          ; переслали коэффициент деления в порт
                        ; делителя частоты обмена
```

Сервис клавиатуры.

BIOS осуществляет преобразование SCAN кода клавиатуры (порядковый номер нажатой клавиши) в код ASCII или расширенный код ASCII. В момент нажатия клавиши выдается запрос на аппаратное прерывание IRQ1 и выполняется соответствующая ему (запросу) программа обработки прерывания, к которой также можно обратиться выдав команду INT 9. Перехватывая обращение к этой подпрограмме, вы можете отслеживать

момент нажатия клавиши и вводить собственную обработку этого прерывания, заменяя или дополняя программу BIOSa, по мере необходимости (см. раздел TSR программы). Подпрограмма ввода с клавиатуры (сервиса клавиатуры) активизируется с помощью функций прерывания INT 16h. Часто в резидентных программах приходится организовывать отклик на нажатие так называемой "горячей клавиши (Hot Key)". Организуется такой отклик, как правило, с помощью перехвата девятого прерывания, хотя в некоторых программах используется проверка буфера клавиатуры и байтов статуса клавиатуры при каждом срабатывании системного таймера (микросхема I8253-5 для XT, и I8254A для AT и более поздних моделей), примерно 18 раз в секунду (18.2 раза или каждые 55 миллисекунд). И в том и в другом случае необходимо учитывать некоторые сведения по строению клавиатуры, расположению и структуре буфера клавиатуры, расположению и структуре слова статуса клавиатуры. Когда нажимается какая-либо клавиша и выдается прерывание IRQ1 клавиатура посылает процессору так называемый SCAN-код этой клавиши, а при отпускании клавиши, клавиатура посылает процессору или тот же самый SCAN-код в котором старший бит установлен в 1 (так работает клавиатура машин класса XT), или неизмененный SCAN-код предваряемый кодом 0F0h, что и служит признаком того, что мы читаем код освобождения клавиши. Чтение клавиатуры осуществляется с помощью обращения к портам 60h и 61h, причем SCAN-код читается из порта 60h, а разряд 7 порта 61h управляет разрешением работы клавиатуры (Enable Keyboard). В разделе посвященном TSR-программам приведен пример программы, обнаруживающей нажатие "горячей клавиши". Ниже представлен фрагмент программы считывающий SCAN-код.

```
        IN      AL, 60h         ; чтение SCAN-кода нажатой клавиши
        CMP     AL, HOT_COD     ; сравнение введенного SCAN-кода с кодом
                                ; клавиши, по которому требуется выполнить
                                ; действие нашей программы ("горячей клавиши")
        JZ      MY_PROG         ; переход на инициализацию программы
                                ; обработки "горячей клавиши"
        JMP     CS:[OLD_KEY]    ; если нажата была не "горячая клавиша" то
                                ; отдаем управление стандартной программе
                                ; обработки прерывания от клавиатуры
MY_PROG:                        ; начало программы обработки "горячей клавиши"
        IN      AL, 61h         ; вводим в AL содержимое порта 61h, в котором
                                ; мы должны установить в 1 старший бит
        MOV     AH, AL          ; сохраняем AL в AH
        OR      AL, 80h         ; установить старший бит регистра AL в 1
```

```
        OUT     61h, AL         ; послать в порт 61h содержимое регистра AL
        MOV     AL, AH          ; восстановили предыдущее содержание в
                                ; регистре AL
        OUT     61h, AL         ; восстановили состояние порта 61h
        MOV     AL, 20h         ; послали в контроллер приоритетных прерываний
                                ; (микросхема I8259A)
        OUT     20h, AL         ; сигнал "конец прерывания"
```

Тело программы

MSDOS

MS DOS - дисковая операционная система, используемая обычно в IBM-совместимых персональных компьютерах. MS DOS - однозадачная, однопользовательская операционная система. Максимальный объем оперативной памяти, доступной программам 640K. Начиная с версии DOS 5.0 драйверы системы и служебные программы DOS могут быть помещены в расширенную оперативную память, если она имеется на компьютере. В более ранних версиях DOS, если объем оперативной памяти в машине превышает 640K то она используется или как виртуальный диск, или так, как это предусмотрено в вашей программе. Некоторые современные пакеты прикладных программ и трансляторы также предусматривают полное использование имеющейся оперативной памяти. Сервис, предусмотренный в MS DOS, позволяет выполнить большинство функций BIOSa, а также, и многие другие функции. Для контроля за использованием оперативной памяти, запуска программ, организации ввода/вывода и многих других целей используются управляющие блоки системы. Некоторые из них создаются в оперативной памяти в момент старта задачи и существуют пока задача работает (PSP, MCB и т.д.), некоторые хранятся на диске и модифицируются по мере необходимости (EXE-header и другие), некоторые создаются при загрузке DOS и хранят актуальную информацию о состоянии системы.

Все системные драйверы DOS хранятся в файле IO.SYS, который загружается первым в оперативную память, а весь сервис DOSa хранится в файле MSDOS.SYS, загружающемся в оперативную память сразу после IO.SYS. Программа, загружающая IO.SYS, довольна проста, поэтому для файла IO.SYS, требуется отводить непрерывное дисковое пространство,

находящееся в самом начале области данных диска (для гибких дисков) и в самом начале области данных раздела жесткого диска. Хотя это и не обязательно, файл MSDOS.SYS также хранится в непрерывной дисковой области, сразу за файлом IO.SYS.

Практически весь сервис DOSa осуществляется через различные функции программного прерывания INT 21h. Номер функции всегда задается в регистре AH, а параметры передаются через остальные регистры. Для некоторых функций через регистр AL передается номер подфункции.

Имеются несколько функций вызываемых через другие прерывания:

- INT 20h - окончить программу (terminate a programm)
- INT 25h - чтение диска (сектор) (absolute disk read)
- INT 26h - запись на диск (сектор) (absolute disk write)
- INT 27h - окончить программу оставив резидентную часть
- INT 28h - разделение времени (возможно использование для мультиплексирования процессов). DOS пользуется этим прерыванием для организации работы команды PRINT в фоновом режиме (DOS timeslice)
- INT 2Eh - выполнить команду DOSa (copy, dir и т.д.) требует наличия доступной памяти для выполнения команды (perform DOS command)
- INT 2Fh - мультиплексирование процессов (multiplex interrupt)
- INT 33h - поддержка мыши (mouse support)
- INT 67h - управление расширенной памятью (Expanded Memory Menager - EMM)

Среди функций DOSa имеются различные функции, выполняющие одно и тоже, но на разном уровне. Те из них, которые работают на более высоком уровне, и, следовательно, удобнее в работе, разработаны в более поздних версиях DOSa. Часть функций DOSa не описана в руководствах фирм MICROSOFT и IBM. Поэтому они часто называются "НЕДОКУМЕНТИРОВАННЫМИ" ("UNDOCUMENTED") или "ВНУТРЕННИМИ" ("INTERNAL" то есть для внутреннего использования в разработках фирмы). Вследствие того, что сведения об этих функциях не даются в официальных документах фирм, производителей операционной системы, они имеют право изменять параметры этих функций и блоков в новых версиях DOSa. Некоторые из наиболее важных недокументированных функций рассмотрены ниже. Многие из недокументированных функций и управляющих блоков DOSa, имеющиеся уже в ранних версиях DOSa, раскрываются при выходе более поздних версий. В таких случаях для функций и блоков даны как общепринятые названия функций и блоков, так и фирменные названия, употребляющиеся в руководствах (например см. [6]).

Система прерываний, перехват прерываний, резидентные программы (TSR-программы).

Для организации системы прерываний отводится первый килобайт оперативной памяти компьютера, где записаны четырехбайтовые вектора прерываний. При выдаче команды INT N вызывается подпрограмма обработки прерывания, адрес которой расположен на N-ном месте в таблице векторов прерываний. Каждый вектор прерывания представляет из себя две двухбайтовые записи, первая из которых - IP программы обработки прерывания (IP - название счетчика команд микропроцессоров семейства 8086, сокращение английского Instruction Pointer), а вторая CS этой программы. Так как, каждый вектор обработки прерывания имеет длину четыре байта то адрес N-го вектора просто равен N*4. Кроме программных прерываний процессор может обрабатывать запросы на прерывания от внешних устройств. Когда внешние устройства выдают запрос на прерывание он поступает на контроллер приоритетных прерываний (микросхема I8259A), который выбирает прерывание с наивысшим приоритетом (если одновременно было получено от внешних устройств несколько запросов на обработку прерываний) и выдает запрос на обработку прерывания непосредственно процессору. Если прерывания разрешены, контроллер приоритетных прерываний выдает на шины процессора команду INT N1, где N1 определяется приоритетом устройства выдавшего запрос и тем как запрограммирован контроллер I8259A. В машинах совместимых с IBM PC принято следующее соответствие:

	аппаратное прерывание		программное прерывание
от системного таймера	IRQ0	-	INT 8
клавиатура	IRQ1	-	INT 9
	IRQ2	-	INT A
	IRQ3	-	INT B
	IRQ4	-	INT C
	IRQ5	-	INT D
НГМД	IRQ6	-	INT E
	IRQ7	-	INT F

Аппаратные прерывания называются также маскируемыми, так как можно запретить (замаскировать) их обработку устанавливая флаг разрешения прерывания IF в 0.

Систему прерываний, установку собственных и перехват системных прерываний часто используют для организации собственной обработки прерываний и расширенной обработки системных прерываний, для контроля текущего состояния и процессов в системе. На перехвате системных и организации собственных прерываний как правило

основана работа так называемых TSR-программ (от Terminate but Stay Resident). При обращении к соответствующим прерываниям управление получает TSR-программа, которая и выполняет нестандартную, или расширенную обработку этого прерывания. Обработка в TSR-программе может быть организована как перед, так и после системной обработки, а также и вместо системной обработки. Все программы обработки прерываний должны оканчиваться командой IRET которая, в отличии от команды RET, не только возвращает управление в вызвавшую программу но и восстанавливает PSW - слово состояния процесса (Process Status Word). Аналогично и команда INT, в отличии от команды CALL, сохраняет в стеке не только адрес возврата но и текущее PSW. Кроме того команда INT сбрасывает флаги IF и TF, а команда IRET, восстанавливая PSW, восстанавливает тем самым и их.

Для перехвата системного и установки своего прерывания обычно используются две функции DOSa (функции 21-го прерывания):

25h - установить вектор прерывания (Set Interrupt Vector);
35h - прочитать вектор прерывания (Get Interrupt Vector).

Однако, можно воспользоваться и прямой адресацией памяти. Второй вариант (прямое считывание и запись в память) может оказаться неработоспособным для мультипрограммных расширений DOSa так как, в них часто используются виртуальные вектора прерываний, а истинные физические адреса векторов остаются неизвестными программе. Оба способа перехвата прерывания проиллюстрированы ниже (перехватывается прерывание 64h):

```
old_vec     dd    ?
.....
process:
.....
            iret
start:
            mov   ax,3564h
      int   21h
      mov   word ptr old_vec,bx
      mov   word ptr old_vec+2,es
      lea   dx,PROCESS
      mov   bx,cs
      mov   ds,bx
      mov   ax,2564h
      int   21h
```

```
old_v       dd    ?
.....
process:
.....
      iret
start:
      xor   ax,ax
      mov   es,ax
      mov   ax,word ptr es:[190h]
      mov bx,word ptr es:[192h]
      mov   word ptr old_v,ax
      mov   word ptr old_v+2,bx
      lea   ax,PROCESS
      mov   bx,cs
      cli
      mov   word ptr es:[190h],ax
      mov   word ptr es:[192h],bx
      sti
```

В примерах программа обработки прерываний называется PROCESS. Если PROCESS перехватывает системное прерывание, то возможно три варианта, проиллюстрированных следующими примерами:

a. Сначала пользовательская обработка прерывания,затем передача управления системной программе обработки прерывания с возвращением управления в программу выдавшую команду INT (программу прерванную для обработки запроса IRQ).

b. Сначала управление передается системной программе обработки прерывания, а затем пользовательская программа осуществляет дополнительную обработку, после чего управление возвращается в прерванную программу.

c. Осуществляется только обработка пользователя.

a.

```
PROCESS:
; ***************
; добавочная ***
; обработка ***
; ***************
 jmp old_vector
```

b.

```
PROCESS:
 pushf
 call old_vector
; ***************
; добавочная ***
; обработка ***
; ***************
 iret
```

c.

```
PROCESS:
; ***************
; обработка ***
; ***************
 iret
```

Содержание добавочной обработки зависит от ваших требований.

Как в программах обработки прерываний, так и в любых других программах требуется уделять повышенное внимание использованию стека. Порча указателя стека, стекового сегмента, несбалансированность команд PUSH и POP приводят к самым непредсказуемым последствиям.

Требования к резидентным программам

Активизация резидентных программ.

При создании программ, остающихся резидентно в памяти необходимо учитывать ряд специфических требований, определяемых особыми условиями, в которых активизируются

и работают эти программы. В самом деле, резидентная программа начинает работать при наступлении какого то, ожидаемого ею события, как правило, прерывая работу выполнявшейся в этот момент программы. Когда резидентная программа оканчивает свою работу она, обычно, должна возвратить управление прерванной программе, причем сохранив в момент прерывания состояние текущего процесса и восстановив его так, чтобы приостановленная на время прерывания программа "не заметила" остановки. Отсюда следует, что к основным требованиям, предъявляемым резидентным программам, должно относится грамотное сохранение состояния прерванного процесса. Следующим требованием является корректное использование сервиса DOSa в резидентном модуле. Вообще говоря, использовать функции DOSa в TSR-программах можно только тогда, когда прерывается выполнение прикладной программы. Если же в момент активизации резидентной программы управление находится у операционной системы, то сервис DOSa использовать нельзя. Это объясняется нереентерабельностью функций DOSa. Таким образом, к основным требованиям на TSR-программы, необходимо также отнести обязательную проверку, можем ли мы в данной ситуации пользоваться теми или иными системными средствами. Как правило, в резидентных программах или вообще отказываются от использования "сомнительных" функций и заменяют их своими, или проверяют состояние флага "нахождения в DOS" (InDOS флаг). Если InDOS флаг не равен 0, то в резидентных программах нельзя пользоваться сервисом DOSa. Если все же необходимо использовать какую либо функцию DOSa, то прибегают к отложенному вызову этих функций. Проверка состояния InDOS флага осуществляется с помощью функции DOSa 34h. Естественно, эта функция может быть использована вне зависимости от состояния InDOS флага, иначе она была бы бессмысленной. Эта функция существовала уже в ранних версиях DOSa, однако, вплоть до версии 5.0 она относилась к числу недокументированных функций. В версии 5.0 она была раскрыта Кроме того, вследствие недокументированности этой функции, она не работает в популярной операционной системе DRDOS (зарегистрированная торговая марка фирмы DIGITAL RESEARCH).

Функция 34h. Получить адрес InDOS флага (Get InDOS Flag Adress)

```
MOV AH, 34h  ; заносим номер функции DOSa в
             ; регистр AH
INT 21h      ; вызов диспетчера DOSa
```

После выполнения этой функции в паре регистров ES:BX находится адрес InDOS флага.

При создании TSR-программ обычно приходиться решать вопрос об условиях её активизации. Заметно выделяется группа программ, активизирующаяся при нажатии горячей клавиши (комбинации клавиш). Часто используются способы активизации по показаниям системного таймера, по обращению к перехваченному прерыванию, по вызову пользовательского прерывания определенного при инсталляции (установке) резидентной программы, которое обслуживается резидентной частью этой программы, и по другим событиям, определенным при разработке программы.

В примерах резидентных программ, приведенных в этом разделе, использованы первые два способа активизации TSR-программ. В примере 8 при одновременном нажатии клавиш ALT и S на экран выводится информационное окно, причем, для иллюстрации работы с функцией DOSa 34h, содержимое и положение этого окна на экране меняется в зависимости от того, находимся мы в состоянии СИСТЕМА или ЗАДАЧА. В примере дана иллюстрация применения механизма локальных меток, впервые введенных в трансляторе фирмы MICROSOFT MASM 5.10.

Пример 8. Пример резидентной программы с использованием HOT KEY

```
CODE_SEG    SEGMENT                 ; определение кодового сегмента
HOT_KEY     PROC    FAR             ; заголовок процедуры
            ASSUME  CS:CODE_SEG     ; определение кодовым сегментам кодовых
                                    ; регистров
            ORG     100h            ; резервируем 256 байт для префикса
                                    ; программного сегмента PSP
BEGIN:                              ; начало кода программы
            JMP     START           ; обход области данных и резидентного модуля
;       область данных
OLD_KEY     LABEL   DWORD           ; определили метку типа двойное слово
                                    ; (этот оператор реально не резервирует места в
                                    ; памяти, а только информирует транслятор о типе
                                    ; метки OLD_KEY)
;       теперь зарезервируем две ячейки памяти по слову каждая
OLD_IP      DW      ?               ; первое слово ячейки OLD_KEY
OLD_CS      DW      ?               ; второе слово ячейки OLD_KEY
;       информационные сообщения, выдаваемые по горячей клавише
INFO1       DB      'состояние СИСТЕМА'
L_INFO1     EQU     $-OFFSET INFO1  ; длина первого сообщения
```



```
INFO2       DB      'состояние ЗАДАЧА'
L_INFO2     =       $-OFFSET INFO2  ; длина второго сообщения
BUFFER      DB      2*L_INFO1 DUP( ? )  ; резервируем место для сохранения
                                    ; изображения находящегося в той области
                                    ; экрана, куда мы будем выводить строку

ATTR1       DB      3Dh             ; атрибут первого окна
ATTR2       DB      6Ah             ; атрибут второго окна
SCREEN      DW      0B000h          ; стартовый адрес монохромного экрана
;       конец области данных

OUT_TEXT    PROC                    ; процедура вывода строки на экран
;       сохраняем в стеке регистры значение которых нам еще понадобится
            PUSH    DI
            PUSH    CX
@@:                                 ; локальная метка (введена в MASM 5.10)
            MOV     AL, DS:[SI]     ; пересылаем строку из
            MOV     ES:[DI], AL     ; ячейки памяти DS:[SI]
            INC     SI              ; в ячейку памяти ES:[DI]
            ADD     DI, 2           ;
            LOOP    @B              ; ссылка на ближайшую локальную метку вверх
                                    ; по программе (вниз-@F)

            POP     CX              ; восстанавливаем регистры
            POP     DI              ; из стека

            RET                     ; возвращаем управление основной программе
OUT_TEXT    ENDP                    ; конец процедуры
PROCESS:                            ; начало резидентного модуля
            PUSH    AX
            IN      AL, 60h         ; занести в AL SCAN-код нажатой клавиши
            CMP     AL, 1Fh         ; 1Fh - SCAN-код клавиши S
            POP     AX
            JZ      VER_STATUS      ; если нажата клавища S, то переходим на
                                    ; проверку байта статуса клавиатуры (ячейка
                                    ; памяти 0:417h)
            JMP     OLD_KEY         ; иначе переходим к оригинальной программе
                                    ; обработки этого прерывания

VER_STATUS:                         ; проверка байта статуса
            PUSH    AX              ; сохраняем регистры,
            PUSH    ES              ; которыми пользуемся
            XOR     AX, AX          ; заносим 0 в регистр AX
            MOV     ES, AX          ; посылаем этот 0 в регистр ES
            MOV     AL, ES:[417h]   ; прочитали байт статуса
            AND     BYTE PTR ES:[417h], 0F7h
;       сбросили бит ALT
            TEST    AL, 8
;       проверили, нажата ли клавиша ALT
            POP     ES              ; восстанавливаем регистры,
            POP     AX              ; которыми пользовались
```

```
        JNZ     OUR_PROG        ; если ALT нажата, то переход на нашу программу
                                ; обработки прерывания
        JMP     OLD_KEY         ; иначе переходим к оригинальной программе
                                ; обработки этого прерывания

OUR_PROG:                       ; сюда мы попадаем, если одновременно нажаты
                                ; клавиши ALT и S
        PUSH    AX              ; сохраняем AX для обработки прерывания
                                ; клавиатуры
        IN      AL, 61h         ; седьмой бит этого порта - разрешение
                                ; клавиатуры (если бит равен нулю) и очистка
                                ; клавиатуры (если бит равен единице)
        MOV     AH, AL          ; сохраняем AL в AH
        OR      AL, 80h         ; устанавливаем старший бит в единицу
        OUT     61h, AL         ; очищаем клавиатуру
        XCHG    AL, AH          ; восстанавливаем AL
        OUT     61h, AL         ; разрешили клавиатуру
        MOV     AL, 20h         ; посылаем контроллеру 8259A
        OUT     20h, AL         ; подтверждение об окончании обработки
                                ; прерывания
        POP     AX              ; восстановили AX

;       сохраняем все регистры, которыми мы будем пользоваться
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    DI
        PUSH    BP
        PUSH    DS
        PUSH    ES
        CLD                     ; устанавливаем направление работы строчных
                                ; команд

;       проверяем, в каком видеорежиме работает видеоадаптер
;       (функция BIOSa 0Fh прерывания INT 10h)
        XOR     DX, DX          ; для монохромных режимов 0 для цветных
                                ; режимов будет 800h
        MOV     AH, 0Fh         ; номер функции занести в регистр AH
        INT     10h             ; вызываем видеосервис BIOSa
;       эта функция возвращает:
;       - в регистре AL - текущее значение видеорежима;
;       - в регистре AH - количество столбцов на экране;
;       - в регистре BH - номер видеостраницы.
;       В нашем примере проверяется только текущее значение видеорежима
        CMP     AL, 7           ; если видеорежим больше 7, то мы в графике,
                                ; и, следовательно, необходимо перейти прямо на
                                ; выход (здесь не предусмотрена работа в графике)
        JNG     @F              ; переход на ближайшую локальную метку вперед
                                ; (локальные метки введены в MASM 5.10)
```



```
        JMP     QUIT            ; на выход
@@:
        JE      MONO            ; переход на работу в монохромном режиме
        CMP     AL, 3           ; находимся ли мы в цветном текстовом режиме
        JG      QUIT            ; на выход
;       устанавливаем сегментные адреса необходимые для работы
;       строковых команд
;       устанавливаем DS на начало видеобуфера
        MOV     DX, 800h        ; для цветного видеорежима

MONO:                           ; переход на эту метку будет если мы находимся
                                ; в монохромном видеорежиме
        ADD     DX, SCREEN      ; сформировали в DX сегментный адрес
                                ; видеобуфера
        MOV     DS, DX          ; установили DS на начало видеобуфера

;       проверим, находимся ли мы в состоянии ЗАДАЧА или СИСТЕМА (функция 34h)
        MOV     AH, 34h         ; номер функции -> AH
        INT     21h             ; вызываем диспетчера DOSa

;       в паре регистров ES:BX возвращается адрес InDos флага.
;       Если он не равен 0, то мы находимся в системе

        CMP     BYTE PTR ES:[BX], 0 ; проверяем InDos флаг
        PUSH    CS              ; заносим в ES то же самое
        POP     ES              ; значение, что и в CS
        JNZ     IN_SYSTEM       ; переходим к выводу INFO1
;       вывод INFO2
        LEA     DI, BUFFER      ; адрес буфера для сохранения содержимого
                                ; экрана
        MOV     SI, 0B00h       ; адрес,откуда сохраняется содержимое экрана
        MOV     CX, L_INFO2     ; длина буфера -> CX
REP     MOVSW                   ; сохраняем в буфере содержимое экрана
;       обменяем содержимое сегментных регистров ES и DS
        PUSH    DS
        PUSH    ES
        POP     DS
        POP     ES
        MOV     CX, L_INFO2     ; длина текста -> CX
        MOV     DI, 0B00h       ; адрес, куда направляется вывод
        MOV     AH, ATTR2       ; занесли атрибут в AH
        MOV     AL, ' '         ; занесли в AL пробел
REP     STOSW                   ; записываем по адресу ES:DI строку пробелов
                                ; с атрибутом для строки INFO2

;       вывод строки INFO2 на экран
        LEA     SI, INFO2       ; адрес INFO2 в SI
        MOV     DI, 0B00h       ; адрес, куда мы выводим в DI
        MOV     CX, L_INFO2     ; длина выводимого текста в CX
        CALL    OUT_TEXT        ; вызываем процедуру вывода текста на экран
```

```
                JMP             SHORT WAITING       ; переход на ожидание нажатия клавиши

IN_SYSTEM:                                          ; вывод строки INFO1 (мы находимся в состоянии
                                                    ; СИСТЕМА)
                LEA             DI, BUFFER          ; адрес буфера для сохранения содержимого
                                                    ; экрана
                MOV             SI, 400h            ; адрес,откуда сохраняется содержимое экрана
                MOV             CX, L_INFO1         ; длина буфера -> CX
        REP     MOVSW                               ; сохраняем в буфере содержимое экрана
;       обменяем содержимое сегментных регистров ES и DS
                PUSH            DS
                PUSH            ES
                POP             DS
                POP             ES
                MOV             CX, L_INFO1         ; длина текста -> CX
                MOV             DI, 400h            ; адрес, куда направляется вывод
                MOV             AH, ATTR1           ; занесли атрибут в AH
                MOV             AL, ' '             ; занесли в AL пробел
        REP     STOSW                               ; записываем по адресу ES:DI строку пробелов
                                                    ; с атрибутом для строки INFO1

;       вывод строки INFO1 на экран
                LEA             SI, INFO1           ; адрес INFO1 в SI
                MOV             DI, 400h            ; адрес, куда мы выводим в DI
                MOV             CX, L_INFO1         ; длина выводимого текста в CX
                CALL            OUT_TEXT            ; вызываем процедуру вывода текста на экран
;       ожидание нажатия клавиши, восстановление экрана
WAITING:
                XOR             AX, AX              ; вызов нулевой функции прерывания
                INT             16h                 ; INT 16h (ждем нажатия какой-либо клавиши)
                LEA             SI, BUFFER          ; адрес буфера в регистр SI восстанавление экрана

;       восстанавливаем все регистры, использовавшиеся в нашей программе
QUIT:
                POP             ES
                POP             DS
                POP             BP
                POP             DI
                POP             SI
                POP             DX
                POP             CX
                POP             BX
                POP             AX
                IRET                                ; конец обработки прерывания


START:                                              ; инсталляция программы
                MOV             AX, 3509h           ; функция DOSa 35h (AH=35h), прочитать
                                                    ; вектор прерывания 9h (AL=09h)
                INT             21h                 ; выполняем функцию
                MOV             OLD_IP, BX          ; сохраняем системный вектор
```



```
                MOV             OLD_CS, ES          ; прерывания в памяти
;       теперь установим свой вектор прерывания INT 09h
                MOV             AX, 2509h           ; функция DOSa 25h (AH=25h), установить
                                                    ; вектор прерывания 9h (AL=09h)
;       оканчиваем программу, оставив резидентную часть
;       в паре регистров DS:DX должен быть адрес программы, которая остается резидентной, причем
;       для DS в нашей программе это требование выполняется автоматически
                LEA             DX, PROCESS         ; занести в регистр DX адрес резидентной
                                                    ; части программы
                INT             21h                 ; вызываем диспетчер DOSa для выполнения
                                                    ; функции
                LEA             DX, START           ; занести в DX адрес первой нерезидентной точки
                                                    ; программы
                INT             27h                 ; выполнить прерывание DOSa "окончить, оставив
                                                    ; резидентную часть" (Terminate but Stay Resident)
HOT_KEY         ENDP                                ; конец процедуры HOT_KEY
CODE_SEG        ENDS                                ; конец сегмента CODE_SEG
                END             BEGIN               ; конец текста программы
```

В примере 9 приведена программа перехватывающая прерывание INT 21h, отслеживающая выполнение функции 4Bh с подфункцией 00 - загрузить и выполнить (LOAD AND EXECUTE) и составляющая файл "истории" (HISTORY), в который записываются все имена стартовавших программ.

Пример 9. Пример резидентной программы "диспетчер"

```
CODE_SEG        SEGMENT                             ; директива определения имени сегмента
INSPECTOR       PROC            FAR                 ; заголовок процедуры
                ASSUME          CS:CODE_SEG         ; директива ассемблера - назначение сегмента ;
CODE_SEG сегментному регистру CS
                ORG             100h                ; резервируем место для PSP (необходимо в COM
                                                    ; программе)
BEGIN:                                              ; начало кода программы
                JMP             START               ; переход на процесс инсталляции (установки) программы
                                                    ; в памяти

;       область данных
OLD_21          LABEL           DWORD               ; определяем ячейку памяти OLD_21 как двойное слово
OLD_IP          DW              ?                   ; определяем первое слово ячейки памяти OLD_21, как
                                                    ; отдельную ячейку с типом слово
OLD_CS          DW              ?                   ; определяем второе слово OLD_21, как самостоя-
                                                    ; тельную переменную

HISTORY         DB              'C:\HISTORY',0      ; определить ASCIIZ (ASCII строку заканчивающуюся
                                                    ; нулем) - путь и имя файла
HANDLE          DW              ?                   ; резервируем место для описателя (HANDLE) файла
L_F     DB      0Dh, 0Ah                            ; коды команд: возврат каретки, перевод строки
```

```
;       ••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••
;       резидентная часть программы
PROCESS:
        CMP     AX, 4B00h       ; проверяем, вызывается ли DOS для выполнения
                                ; функции "ЗАГРУЗИТЬ И ВЫПОЛНИТЬ"
                                ; (LOAD AND EXECUTE)
        JZ      MY_PROG         ; если да - перейти на нашу программу
        JMP     OLD_21          ; если нет - отдать управление диспетчеру DOSa

MY_PROG:                        ; модуль, записывающий имя стартующей программы, в
                                ; файл HISTORY

; сохраним в стеке все используемые в резидентном модуле регистры
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    DI
        PUSH    BP
        PUSH    DS
        PUSH    ES

;       пытаемся открыть файл HISTORY
;       для выполнения функции OPEN нужны регистры DS:DX, также, как и для функции LOAD AND
;       EXECUTE, следовательно, сохраняем их
        PUSH    DS
        PUSH    DX
;       в регистре DS должно быть то же значение, что и в CS
        MOV     DX, CS
        MOV     DS, DX

        MOV     AH, 3Dh         ; номер функции OPEN занесли в регистр AH
        MOV     AL, 01          ; занести в AL режим работы с файлом (0 - только
                                ; чтение, 1 - только запись, 2 - чтение и запись)
        LEA     DX, HISTORY     ; занести в DX адрес пути
        INT     21h             ; вызов диспетчера DOSa

        JC      CREATE_FILE ; если при открытии файла произошла ошибка, то
                                ; делаем попытку создать файл
;       если все в порядке, то устанавливаем указатель файла на конец файла
        MOV     HANDLE, AX  ; так как открытие прошло нормально, то в AX находит-
                                ; ся описатель файла. Сохраняем его в памяти
;       используем функцию 42h - прочитать/установить указатель файла
        MOV     BX, HANDLE  ; занести описатель файла в BX
        MOV     AL, 2           ; сдвигать указатель файла:
                                ; 0 - от начала;
                                ; 1 - от текущей позиции;
                                ; 2 - от конца файла.
        XOR     CX, CX          ; указатель сдвигается на
```



```
        MOV     DX, CX      ; расстояние CX*65536 + DX, то есть в нашем случае
                                ; на 0
        MOV     AH, 42h         ; номер функции в AH
        INT     21h             ; вызов диспетчера DOSa
        POP     DX              ; восстанавливаем
        POP     DS              ; регистры DX и DS
        JMP     WRITE_FILE  ; переходим к дополнению файла HISTORY
CREATE_FILE:                    ; в случае неудачи открытия файла делается попытка
                                ; создать файл (функция 3Ch)
        MOV     AH, 3Ch         ; номер функции в AH
        LEA     DX, HISTORY     ; адрес пути в паре DS:DX
        MOV     CX, 0           ; атрибут файла
        INT     21h             ; вызов диспетчера DOSa
        POP     DX              ; восстанавливаем
        POP     DS              ; регистры DX и DS
        JC      QUIT_RESIDENT       ; если попытка создать файл не удалась, то
; переход на конец резидентного модуля
        MOV     HANDLE, AX  ; сохраняем описатель файла в памяти
WRITE_FILE:                     ; блок записи имени задачи в файл HISTORY
;       определим длину имени задачи с помощью команды SCASB. При этом используем тот факт, что
;       при вызове функции 4B00h имя пути - ASCIIZ строка по адресу DS:DX
        MOV     AX, DS          ; подготавливаем работу SCASB
        MOV     ES, AX          ; для этого помещаем в ES
        MOV     DI, DX          ; содержимое DS, а в регистр DI - содержимое DX
        MOV     AL, 0           ; в строке имени ищем 0
        MOV     CX, 40h         ; в регистр CX занесли максимальную возможную
                                ; длину имени файла
        REPNE SCASB             ; ищем 0 - определяем длину файла
        SUB     CX, 40h         ; длина имени = -(CX - 40h)
;       запишем имя задачи в файл HISTORY
        MOV     BX, HANDLE  ; в BX описатель файла
        MOV     AH, 40h         ; в AH номер функции
;       в регистре CX уже находится длина записи
;       в паре DS:DX уже находится адрес записи
        INT     21h             ; вызов диспетчера DOSa
;       запишем в файл перевод строки
        MOV     CX, 2           ; длина записи
        PUSH    CS              ; в паре регистров
        POP     DS              ; DS:DX запись
        LEA     DX, L_F         ; перевода строки
        MOV     BX, HANDLE  ; описатель файла в BX
        MOV     AH, 40h         ; номер функции
        INT     21h             ; вызов диспетчера DOSa
;       закрываем файл HISTORY
        MOV     BX, HANDLE  ; описатель файла в BX
        MOV     AH, 3Eh         ; функция закрытия файла
        INT     21h             ; вызов диспетчера DOSa

QUIT_RESIDENT:          ; конец резидентного модуля
;       восстанавливаем все регистры, сохраненные в начале резидентного модуля
        POP     ES
```

```
                POP     DS
                POP     BP
                POP     DI
                POP     SI
                POP     DX
                POP     CX
                POP     BX
                POP     AX
                JMP     OLD_21          ; передаем дальнейшую обработку этой функции DOS
;       конец резидентной части
                ASSUME  DS:CODE_SEG     ; директива ассемблера - назначение сегмента
                                        ; CODE_SEG сегментному регистру DS
START:                                  ; инсталляция резидентной программы
;       прочитаем вектор прерывания INT 21h
                MOV     AX, 3521h       ; функция 35h - читать вектор прерывания, номер
                                        ; вектора, который мы читаем - 21h
                INT     21h             ; вызов диспетчера DOSa
;       после выполнения этой функции в регистре BX - IP вектора, а в регистре ES - CS
                MOV     OLD_IP, BX      ; сохраняем счетчик команд оригинального вектора
                                        ; прерывания в ячейке памяти
                MOV     OLD_CS, ES      ; сохраняем сегмент оригинального вектора
                                        ; прерывания в ячейке памяти

;       установим в качестве нового адреса прерывания адрес нашего резидентного модуля
                MOV     AX, 2521h       ; функция 25h - установить вектор прерывания 21h
                LEA     DX, PROCESS     ; записать в DX адрес нашего резидентного модуля
                                        ; сегмент нашего резидента должен быть в DS, что в этой
                                        ; программе выполняется
                INT     21h             ; вызов диспетчера DOSa

;       окончить программу, оставив резидентную часть (функция DOSa 31h)
;       параметры передаваемые функции:
;       номер функции 31h - в регистре AH;
;       код возврата - в регистре AL;
;       размер памяти отводимой резиденту (в 16-и байтовых параграфах) - в регистре DX.
                MOV     AX, 3100h       ; номер функции - 31h, код возврата - 0
                LEA     DX, START       ; занести в DX адрес конца резидентной части, то есть,
                                        ; для этой программы (COM), размер резидентной части
                                        ; в байтах
                MOV     CL, 4           ; счетчик сдвига - необходим для преобразования
                                        ; размера резидентной части в параграфы (деления на
                                        ; 16 путем сдвига)
                SHR     DX, CL          ; преобразовали к параграфам сдвинув вправо на 4
                                        ; разряда
                INC     DX              ; добавили к размеру 1, учтя тем самым, ошибку
                                        ; деления
                INT     21h             ; выполним функцию - окончить, оставив резидентную
                                        ; часть
INSPECTOR       ENDP
CODE_SEG        ENDS
                END     BEGIN
```


Функция DOSa 52h

Одной из важнейших, если не важнейшей, из недокументированных, является функция DOSa 52h, которая дает программе большое количество разнообразной информации о состоянии системы, системных блоках, наличии драйверов различных устройств и многом другом. Полный разбор информации, предоставляемой программе этой функцией не входит в задачу автора. Однако, некоторые, наверное наиболее важные сведения, будут рассмотрены.

Вызов функции 52h

```
MOV     AH, 52h     ; номер функции в регистр AH
INT     21h         ; вызов диспетчера DOSa для выполнения
                    ; функции 52h
```

Эта функция, возвращает в паре регистров ES:BX адрес "списка списков" (List of Lists), или, как говорится в популярной программе "Tech Help", "блок переменных DOSa" (Dos Vars Block). Ниже приведена структура этого списка для версий DOSa начиная с 3.30.

АДРЕС	СОДЕРЖИМОЕ ПОЛЯ
ES:[BX-2]	Сегмент первого блока управления распределением памяти. Этот блок широко известен под названием Memory Control Block, или MCB. Вплоть до версии DOSa 5.0 относился к недокументированным. В версии 5.0 блок раскрыт. Фирменное название - Memory Arena
ES:[BX]	Указатель (4 байта - тип дальний) на первый блок параметров диска DPB или (Disk Parameter Block).
ES:[BX+4]	Указатель на первую таблицу открытых файлов.
ES:[BX+8]	Указатель на начало драйвера часов (CLOCK$ device driver).
ES:[BX+0Ch]	Указатель на начало драйвера основной консоли (CON: device driver).
ES:[BX+10h]	Это поле содержит максимальный размер сектора, поддерживаемый системой.
ES:[BX+12h]	Указатель на буфер сектора.
ES:[BX+16h]	Указатель на информационную таблицу путей к устройствам.
ES:[BX+1Ah]	Указатель на таблицу FCB. Действует только в том случае, когда в файле CONFIG.SYS указан параметр "FCBS=".
ES:[BX+1Eh]	Размер таблицы FCB.
ES:[BX+20h]	Содержит число логических устройств, представленных в системе.
ES:[BX+21h]	Значение параметра "LASTDRIVE=" из CONFIG.SYS
ES:[BX+22h]	Начало кода (не указатель) первого драйвера системы - NUL device driver

Описание управляющих блоков, упомянутых выше, дано в разделе "Управляющие блоки системы".

Управляющие блоки DOSa.

M C B (Memory Arena)

MCB - MEMORY CONTROL BLOCK, блок управления распределением памяти. Все распределение памяти в операционной системе полностью управляется цепочкой этих блоков, где каждый из предыдущих блоков указывает на следующий. До появления DOSa версии 5.0 он являлся недокументированным. Этот блок позволяет программе составить полную карту использования оперативной памяти. (см. пример 10 в конце раздела).

Структура MCB (Для MS-DOS версий до 3.30 включительно)

смещение		длина	содержание поля
0	DB	байт	сигнатура: 4Dh - не последний 5Ah - последний блок
1	DW	слово	сегмент владельца MCB
3	DW	слово	смещение в параграфах от конца текущего до начала следующего MCB (длина параграфа 16 байт)
5	DB	11 байт	резерв

Структура MCB (Для MS-DOS версий от 4.0 и старше)

смещение		длина	содержание поля
0	DB	байт	сигнатура: 4Dh - не последний 5Ah - последний блок
1	DW	слово	сегмент владельца MCB
3	DW	слово	смещение в параграфах от конца текущего до начала следующего MCB (длина параграфа 16 байт)
5	DB	3 байта	резерв
8	DB	8 байт	имя владельца

DISK PARAMETER BLOCK - содержит информацию обо всех блочных устройствах системы. Подобно блоку MCB каждый предыдущий DPB, кроме последнего, ссылается на следующий.

Блочному драйверу, то есть драйверу обслуживающему устройства передающие данные блоками, а не отдельными символами, может принадлежать несколько DPB.

Структура DPB (DOS 3.X).

смещение		длина	содержание поля
0	DB	байт	Номер первого устройства, обслуживаемого драйвером.
1	DB	байт	Номер логического устройства, которое описывается этим DPB. Если это и предыдущее поле содержат 0FFh, то данный DPB - ошибочный.
2	DW	слово	Размер логического сектора устройства, измеренный в байтах
4	DB	байт	Количество секторов в кластере, уменьшенное на 1. Количество секторов в кластере должно быть степенью 2.
5	DB	байт	Показывает, на сколько разрядов нужно сдвинуть количество кластеров влево, чтобы получить количество секторов.
6	DW	слово	Количество резервных секторов на диске. Показывает количество резервных секторов от начала физического диска до начала логического устройства. Резервные сектора могут содержать информацию о делении физического устройства на логические (таблица разделов жесткого диска - Disk Partition Table)
8	DB	байт	Количество копий FATa. Обычно 2.
9	DW	слово	Количество точек входа в корневой директории (Root Directory Entries)
0Bh	DW	слово	Первый сектор данных.
0Dh	DW	слово	Номер последнего кластера. Если этот номер меньше чем 0FF6h, то используется 12-битовый FAT, иначе используется 16-битовый FAT.
0Fh	DB	байт	Размер FATa в логических секторах.
10h	DW	слово	Первый сектор, содержащий точки входа в корневую директорию.
12h	DD	двойное слово	Указатель на драйвер, которому принадлежит этот блок.
16h	DB	байт	Описатель носителя (MEDIA DESCRIPTOR).
17h	DB	байт	Флаг доступа. Доступ к устройству разрешен, если этот флаг равен нулю. Если этот байт равен 0FFh, то блок может быть перестроен
18h	DD	двойное слово	Указатель на следующий DPB. Если это поле содержит 0FFFFh, то этот DPB - последний

В структуре DPB для DOSa версий 4.x и 5.x, начиная с адреса 0Fh блок DPB имеет следующий вид:

0Fh	DW	слово	Размер FATa в логических секторах (поле изменено в связи с тем, что максимальный раздел жесткого диска теперь превышает 32 мегабайта)
10h	DW	слово	Первый сектор, содержащий точки входа в корневую директорию.
12h	DD	двойное слово	Указатель на драйвер, которому принадлежит этот блок.
16h	DB	байт	Описатель носителя (MEDIA DESCRIPTOR).
17h	DB	байт	Флаг доступа. Доступ к устройству разрешен, если этот флаг равен нулю. Если этот байт равен 0FFh, то блок может быть перестроен
18h	DD	двойное слово	Указатель на следующий DPB. Если это поле содержит 0FFFFh, то этот DPB - последний

Таблица открытых файлов (Open File Table)

Open file table - содержит сведения обо всех открытых в данный момент файлах.

Структура таблицы открытых файлов (DOS 3.x)

смещение		длина	содержание поля
0	DD	двойное слово	указатель на следующую таблицу в списке. Первое слово этого указателя (смещение) равно 0FFFFh в том случае, если таблица - последняя в списке
4	DW	слово	Число входов (Entries) в таблице. Длина каждого входа - 53 байта (35h байтов).
6			Начало первого входа (описателя) таблицы

Структура одного элемента (входа, описателя, Entry) таблицы открытых файлов (DOS 3.x)

смещение		длина	содержание поля
0	DW	слово	Число описателей файла (File handlles), ссылающихся на этот элемент таблицы открытых файлов.
2	DW	слово	Режим открытия файла. Бит 15 установлен в 1, если файл открыт через FCB.
4	DB	байт	Атрибут файла
5	DW	слово	Слово информации об устройстве. (Такое же, как в функции DOSa 44h с подфункцией 0. Когда бит 14 установлен, это значит, что не меняется дата/время последнего обновления файла)
7	DD	двойное слово	Указатель на заголовок драйвера устройства (для символьных драйверов), или указатель на блок параметров диска (DPB), если драйвер блочный.

0Bh	DW	слово	Стартовый кластер файла.
0Dh	DW	слово	Время последнего обновления файла (В упакованном формате, таком же, как в функции DOSa 57h)
0Fh	DW	слово	Дата последнего обновления файла (В упакованном формате)
11h	DD	двойное слово	Размер файла
15h	DD	двойное слово	Текущее положение указателя файла
19h	DW	слово	Относительный номер последнего прочитанного кластера внутри файла
1Bh	DW	слово	Абсолютный номер последнего прочитанного кластера
1Dh	DW	слово	Номер сектора, содержащего DIRECTORY ENTRY этого файла
1Fh	DB	слово	Номер DIRECTORY ENTRY этого файла внутри сектора
20h	DB	11 байт	Имя файла. Записано в виде: имя + расширение, причем расширение не отделено точкой от имени.
2Bh	DD	двойное слово	?
2Fh	DW	слово	Номер сетевой машины для которой открыт файл (SHARE.EXE)
31h	DW	слово	Сегмент PSP владельца файла
33h	DW	слово	?

Структура таблицы открытых файлов (DOS 4.x)

смещение		длина	содержание поля
0	DD	двойное слово	указатель на следующую таблицу в списке. Первое слово этого указателя (смещение) равно 0FFFFh в том случае, если таблица - последняя в списке
4	DW	слово	Число входов (Entries) в таблице. Длина каждого входа - 59 байтов (3Bh байтов).
6			Начало первого входа (описателя) таблицы

Начиная со смещения 1Bh в DOS 4.x и 5.x элемент таблицы открытых файлов выглядит так:

1Bh	DW	двойное слово	Номер сектора, содержащего DIRECTORY ENTRY этого файла
1Fh	DB	слово	Номер DIRECTORY ENTRY этого файла внутри сектора
20h	DB	11 байт	Имя файла. Записано в виде: имя + расширение, причем расширение не отделено точкой от имени.
2Bh	DD	двойное слово	?
2Fh	DW	слово	Номер сетевой машины для которой открыт файл (SHARE.EXE)
31h	DW	слово	Сегмент PSP владельца файла
33h	DW	слово	?
35h	DW	слово	Абсолютный номер последнего прочитанного кластера
37h	DD	двойное слово	?

Заголовок буфера сектора

Функция DOSa 52h содержит указатель на первый буфер, предназначенный для хранения сектора в дисковых операциях ввода/вывода. Каждый такой буфер предваряется заголовком буфера, содержащим информацию о том, с каким устройством производятся для данного буфера операции ввода/вывода, какого типа данные находятся в буфере, где искать следующий буфер сектора и т.д.. Количество буферов сектора в системе определяется параметром "BUFFERS=XX" файла CONFIG.SYS.

Структура заголовка буфера сектора

смещение		длина	содержание поля
0	DD	двойное слово	Указатель на следующий буфер сектора. Для последнего буфера в этом поле должно быть -1 (0FFFFFFFFh)
4	DB	байт	Номер устройства, для которого через этот буфер производится ввод/вывод. Если буфер не используется, значение этого байта - 0FFh
5	DB	байт	Флаг типа данных. Биты в этом поле обозначают: 1 - данные типа "таблица размещения файлов" (FAT); 2 - тип данных - каталог или подкаталог (Directory or Subdirectory); 3 - файл.
6	DW	слово	Номер логического сектора.
8	DW	слово	?
0Ah	DD	двойное слово	Указатель на блок параметров диска.
0Eh	DW	слово	Поле не используется.

Таблица путей устройств

Таблица путей устройств содержит информацию для всех устройств системы о том, где, в каком каталоге, вы будете находится, при переключении на то или иное конкретное устройство.

Структура таблицы путей устройств

смещение		длина	содержание поля
0	DB	64 байта	Полное имя файла в форме ASCIIZ строки (имя и путь).
40h	DD	двойное слово	Резерв. Установлен 0.
44h	DB	байт	Флаг. Все кроме последнего элемента содержат в этом поле 40h. Последний - содержит 0.
45h	DD	двойное слово	Указатель на блок параметров диска.
49h	DW	слово	Текущий номер блока.
4Bh	DD	двойное слово	Указатель. ?
4Fh	DW	слово	?

P S P

PROGRAMM SEGMENT PREFIX - предшествует любой программе. Содержит информацию необходимую для корректного исполнения и завершения программы. Блок PSP создается в оперативной памяти в момент запуска программы. PSP всегда начинается в начале сегмента со смещением 0. Поскольку программы типа COM занимают ровно один сегмент, то PSP непосредственно предшествует программе. Длина PSP 256 байт, поэтому в программах типа COM требуется псевдооператор ORG 100h, резервирующий место для PSP. В программах любого типа, как EXE, так и COM, адрес PSP в момент старта DS:0000.

Структура PSP.

смещение		длина	содержание поля
0	DW	слово	код команды INT 20h
2	DW	слово	общий размер памяти машины
4	DB	байт	резерв
5	DB	5 байт	код команды CALL FAR к диспетчеру DOSa
0Ah	DW	слово	адрес окончания программы
0Ch	DW	слово	сегмент окончания программы
0Eh	DW	слово	адрес обработки cntrl break
10h	DW	слово	сегмент обработки cntrl break
12h	DW	слово	адрес программы обработки критических ошибок
14h	DW	слово	сегмент программы обработки критических ошибок
16h	DW	слово	сегмент PSP родительского процесса
18h	DB	20 байт	резерв

2Ch	DW	2 байта	сегмент ENVIRONMENT
2Eh	DW	2 слова	старое содержимое SS:SP
32h	DW	2 байта	MAX количество открытых файлов
34h	DW	слово	адрес таблицы открытых файлов
36h	DW	слово	сегмент таблицы открытых файлов
38h	DW	24 байта	не используется
50h	DB	3 байта	коды команд: INT 21h, RETF
53h	DW	слово	не используется
55h	DB	7 байт	расширение FCB1
5Ch	DB	9 байт	FCB1
65h	DB	7 байт	расширение FCB2
6Ch	DB	9 байт	FCB2
80h	DB	1 байт	длина командной строки
81h	DB	127 байт	командная строка

Dos Environment (Окружение DOSa, расширение DOSa)

DOS Environment - символьная строка, создающаяся для каждой программы в оперативной памяти и содержащая сведения из файлов CONFIG.SYS и AUTOEXEC.BAT в форме набора из ASCIIZ строк (символьных строк оканчивающихся нулем) ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ. Первая строчка, всегда имеющаяся в DOS Environment, указывает местонахождение командного интерпретатора - программы COMMAND.COM, и имеет вид "COMSPEC=путь к файлу COMMAND.COM". Все остальные подстроки заполняются командами "PATH=", "PROMPT","SET имя=". Так, например, если файл CONFIG.SYS содержит строку:

SHELL=D:\DOS\COMMAND.COM,

а файл AUTOEXEC.BAT содержит строки:

```
PATH=C:\DOS;C:\;D:\USER;D:\;
PROMPT $p$g
SET HELPPATH=C:\THELP
SET TMP=C:\TEMP,
```

то DOS Environment будет выглядеть так (изображено в виде строки ассемблера, резервирующей область памяти):

```
DOS_Environment   DB   'COMSPEC=D:\DOS\COMMAND.COM',0
                  DB   'PATH=C:\DOS;C:\;D:\USER;D:\;',0
                  DB   'PROMPT $p$g',0
                  DB   'HELPPATH=C:\THELP',0
                  DB   'TMP=C:\TEMP',0
```


Кроме того, для DOSa версий старше 3.0, после последнего из этих полей располагается поле, содержащее полный путь и имя программы, владельца данного окружения DOSa (DOS Environment). Это поле отделено от остальных четырехбайтовой последовательностью:

DB 0,0,1,0.

Таким образом полностью, для программы DIGGER.COM, наш пример будет выглядеть так:

```
DOS_Environment   DB   'COMSPEC=D:\DOS\COMMAND.COM',0
                  DB   'PATH=C:\DOS;C:\;D:\USER;D:\;',0
                  DB   'PROMPT $p$g',0
                  DB   'HELPPATH=C:\THELP',0
                  DB   'TMP=C:\TEMP',0
                  DB   0
                  DB   1
                  DB   0
                  DB   'D:\GAME\DIGGER.COM',0
```

EXE - HEADER

EXE - HEADER, заголовок файлов типа EXE. Создается редактором связей и записывается на диск непосредственно перед файлом типа EXE. Используется для настройки программы типа EXE, по конкретным адресам памяти, непосредственно перед стартом.

Структура EXE - HEADERa

смещение		длина	содержание поля
0	DB	2 байта	4D5Ah - сигнатура, служащая для идентификации EXE файлов
2	DW	слово	число байтов в последнем блоке
4	DW	слово	размер файла в 512 байтовых блоках (включая HEADER)
6	DW	слово	число элементов таблицы перемещений
8	DW	слово	размер HEADERa в параграфах
0Ah	DW	слово	минимальное количество требующейся для размещения программы памяти
0Ch	DW	слово	загрузка в младшие (0FFFFh) или в старшие (0h) адреса памяти. Если не 0 и не 0FFFFh, то тоже самое, что и предыдущее поле, но максимальное

0Eh	DW	слово	смещение стекового сегмента (SS) относительно исполняемого модуля
10h	DW	слово	значение SP в момент старта
12h	DW	слово	контрольная сумма слов в файле. Обычно не вырабатывается, не используется и не проверяется
14h	DW	слово	значение счетчика команд IP в момент старта программы
16h	DW	слово	смещение сегмента команд CS относительно исполняемого модуля
18h	DW	слово	смещение первого элемента таблицы перемещений
1Ah	DW	слово	номер оверлея. 0 - если основная программа
1Ch			начало таблицы перемещений

Редакторы связей фирмы "BORLAND" (TLINK) позволяют создавать EXE-Header с размером некратным 512 байт.

Представляет интерес написать программу, позволяющую определить содержимое оперативной памяти, так как существуют программы несовместимые друг с другом. Кроме того, грамотно написанная резидентная программа, чтобы невозможно было случайно запустить ее многократно, должна была бы отслеживать: не было ли раньше запусков этой программы? Для решения этих задач можно использовать такой подход:

1. выполнить функцию DOSa 52h;
2. прочитать в ячейке памяти ES:[BX-2] сегмент первого блока управления памятью;
3. проверить является ли владельцем этого блока программа (признаком программы является то, что сегмент владельца начинается с PSP);
4. прочитать в PSP адрес DOS Environment;
5. просканировать DOS Environment и определить имя владельца;
6. если блок управления памятью последний, закончить эту часть программы;
7. если MCB не последний, найти следующий и снова уйти на пункт 3.

Пример 10 - подпрограмма, определяющая собственное наличие в оперативной памяти. В случае, если один экземпляр программы уже есть в оперативной памяти, подпрограмма возвращает флаг переполнения CF=1. Предполагается, что эта подпрограмма транслируется вместе с вызывающей программой.

Пример 10. Определение собственного наличия в памяти

```
SELF_SEARCH     PROC                    ; заголовок процедуры
                MOV   CS:OWN_PSP, CS    ; предполагая, что в CS находится адрес
                                        ; собственного PSP нашей программы (верно
                                        ; для программ типа COM), сохраняем этот
                                        ; адрес в ячейке оперативной памяти
                JMP   READ_SELF_NAME    ; обход области данных подпрограммы
SELF_NAME       DB    40h DUP(?)        ; буфер для хранения собственного имени
                                        ; программы
NAME_SHIFT      DW    ?                 ; смещение имени программы от начала DOS
                                        ; Environment до начала имени программы
NAME_LENGTH     DW    ?                 ; длина собственного имени программы
OWN_PSP         DW    ?                 ; сегмент собственного PSP программы

;       сначала считываем в буфер собственное имя нашей программы
READ_SELF_NAME:
                MOV   ES, CS:[2Ch]      ; помещаем в сегментный регистр ES адрес
                                        ; Environment нашей программы
                MOV   AL, 0             ; 0 -> регистр AL
                MOV   CX, 400h          ; счетчик -> CX
                XOR   DI, DI            ; очищаем DI
NEXT_SEARCH:
        REPNE SCASB                     ; ищем 0 в Environment

;       проверим, следует ли за этим нулем признак начала программы
                CMP   WORD PTR ES:[DI], 100h
                JNZ   NEXT_SEARCH       ; если нет, ищем следующий 0
                ADD   DI, 3             ; если да, то устанавливаем
                                        ; регистр DI на начало имени
                MOV   NAME_SHIFT, DI    ; записываем смещение от начала Environment до
                                        ; начала имени в память
        REPNE SCASB                     ; ищем конец имени
                DEC   DI
                SUB   DI, NAME_SHIFT
                MOV   CX, DI            ; вычисляем длину имени
                MOV   SI, NAME_SHIFT    ; подготовка SI для MOVSB
                LEA   DI, SELF_NAME     ; подготовка DI для MOVSB
                MOV   NAME_LENGTH, CX   ; сохраняем длину имени
                PUSH  DS                ; устанавливаем сегментные
                PUSH  ES                ; регистры ES и DS для исполь-
                POP   DS                ; зования в команде MOVSB
                POP   ES
        REP     MOVSB                   ; считываем собственное имя программы
                MOV   AH, 52h           ; вызываем функцию DOSa 52h -
                INT   21h               ; прочитать системную информацию
                MOV   ES, WORD PTR ES:[BX-2]    ; прочитать адрес первого MCB
SEARCH_MCB:
                CALL  SEARCH_NEXT_MCB           ; вызываем подпрограмму поиска
                                                ; следующего MCB
                JC    QUIT              ; если MCB последний, то идем на выход
```

```
                CALL    CHECK_PSP       ; вызываем подпрограмму, проверяющую,
                                        ; является ли владельцем блока памяти PSP
                JC      SEARCH_MCB      ; если нет, то поиск следующего MCB
                CALL    CMP_NAME        ; вызываем подпрограмму, сравнивающую имя
                                        ; нашей программы с именем владельца этого
                                        ; блока
                JC      SEARCH_MCB      ; при несовпадении имен переход на поиск
                                        ; следующего MCB
QUIT:
                CMC                     ; устанавливаем CF для передачи информации
                                        ; вызывающей программе
                RET                     ; возвращаемся в вызывающую программу

SEARCH_NEXT_MCB PROC
                MOV     BX, ES          ; посылаем в BX адрес MCB
                ADD     BX, ES:[3]      ; добавляем количество зарезервированной этим
                                        ; блоком памяти
                INC     BX              ; добавляем 1 параграф, занимаемый самим
                                        ; блоком
                MOV     ES, BX          ; посылаем адрес MCB в ES
                CMP     BYTE PTR ES:[0], 5Ah ; проверяем - последний ли это блок:
                JNZ     NOT_LAST        ; если нет - выход со сбросом флага CF,
                STC                     ; если последний, то устанавливаем флаг CF
                RET                     ; и выходим
NOT_LAST:                               ; выход со сбросом CF
                CLC
                RET
SEARCH_NEXT_MCB ENDP                    ; конец подпрограммы
                                        ; поиска MCB

CHECK_PSP       PROC                    ; процедура проверки - владелец блока памяти
                                        ; - PSP?
                MOV     DS, ES:[1]      ; посылаем адрес владельца в регистр ES
                CMP     WORD PTR DS:[0], 20CDh      ; проверяем содержат ли первые два байта
                                                    ; владельца код INT 20h
                JNZ     NOT_PSP         ; переход, если владелец блока не PSP
                CLC                     ; очистка флага CF
                RET                     ; возврат в вызывающую подпрограмму в
                                        ; случае, если владелец блока - PSP
NOT_PSP:
                STC                     ; установка флага CF
                RET                     ; возврат в вызывающую подпрограмму в
                                        ; случае, если владелец блока - не PSP
CHECK_PSP       ENDP                    ; конец подпрограммы

CMP_NAME        PROC                    ; подпрограмма проверки имени
;       следующие четыре оператора служат для проверки: не принадлежит ли этот блок управления
;       нашей программе
                MOV     AX, DS          ; посылаем DS в AX
                CMP     AX, OWN_PSP     ; проверяем: не наша ли программа владелец
                                        ; этого блока памяти
```



```
                JNZ     NOT_OWN_PSP     ; если владелец блока не наша программа, то
                                        ; переход на сравнение имен
                STC                     ; устанавливаем флаг CF
                RET                     ; выход из подпрограммы

NOT_OWN_PSP:                            ; сравнение имен нашей программы и владельца
                                        ; блока памяти
                PUSH    DS              ; сохранение DS
                PUSH    ES              ; сохранение ES
                MOV     ES, DS:[2Ch]    ; записываем в ES сегмент DOS Environment
                                        ; владельца блока памяти (MCB)
                PUSH    CS              ; через стек записываем содержимое
                POP     DS              ; регистра CS в DS
                MOV     CX, NAME_LENGTH ; посылаем в счетчик цикла (регистр CX) длину
                                        ; имени нашей программы
                INC     CX              ; увеличиваем счетчик на 1, чтобы учесть 0,
                                        ; оканчивающий имя программы
                LEA     SI, SELF_NAME   ; помещаем в SI адрес имени нашей программы
                MOV     DI, NAME_SHIFT  ; посылаем в DI смещение от начала DOS
                                        ; Environment до начала имени программы
                CMP     AX, AX          ; устанавливаем флаг ZF (необходимо для
                                        ; работы префикса REPE)
                REPE    CMPSB           ; сравниваем строчки
                JNZ     NOT_SAME_NAME   ; если они не равны переходим на выход с
                                        ; установкой CF
                CLC                     ; если имена совпадают, сбрасываем флаг
                                        ; CF - сигнал вызвавшей подпрограмме
                JMP     QUIT_CMP_NAME   ; идем на выход
NOT_SAME_NAME:
                STC                     ; установили CF - все хорошо

QUIT_CMP_NAME:
;       восстанавливаем сегментные регистры
                POP     ES
                POP     DS
                RET                     ; выход из подпрограммы
CMP_NAME        ENDP                    ; конец подпрограммы
SELF_SEARCH     ENDP
```

Загружаемые драйверы устройств

Драйвер - это специальным образом оформленная программа, обслуживающая какое либо устройство ввода/вывода. В MS-DOS различаются два основных типа драйверов - стандартные и загружаемые. Стандартные драйверы содержатся в файле IO.SYS и активизируются системой во время начальной загрузки. Очень мощным и важным средством DOSa является возможность подключать драйверы обслуживающие нестандартные устройства (например сканнер) и обращаться к ним используя обычные функции DOSa работы с

файлами. Для подключения этих драйверов к системе необходимо использовать в файле конфигурации системы CONFIG.SYS строчку вида "DEVICE=полное имя драйвера". Загружаемые драйверы бывают двух типов:

1. Символьные драйверы, драйверы в которых обмен данными с устройством производится побайтно (посимвольно). Типичным символьным драйвером является драйвер консоли.
2. Блочные драйверы - в них обмен данными с устройством производится блоками. Типичным примером блочного драйвера является драйвер диска.

Программы загружаемых драйверов должны иметь строго определенную структуру и состоять из трех основных частей. Первая из этих частей называется ЗАГОЛОВОК ДРАЙВЕРА и информирует систему о типе и возможностях загружаемого драйвера. Смещение (OFFSET) заголовка драйвера от начала кода драйвера всегда должно быть равно нулю.

Следующая обязательная часть драйвера называется обычно ПРОЦЕДУРА СТРАТЕГИИ или СТРАТЕГИЯ и служит для передачи драйверу команды и данных, а также для получения от драйвера кода возврата и данных.

Последняя обязательная часть драйвера называется обычно ПРОЦЕДУРА ПРЕРЫВАНИЯ или ПРЕРЫВАНИЕ, как предлагается в фирменной документации [6,7], или, иногда, КОМАНДА. Это собственно и есть та часть, которая получает и обрабатывает команды системы, передает данные устройству и управляет устройством, получает данные от устройства и вырабатывает коды возврата. В отличии от заголовка драйвера и процедуры стратегии, процедура прерывания не имеет строго определенной формы, однако требует соблюдения определенных соглашений о связях. Разберем подробно все три элемента загружаемого драйвера.

Заголовок драйвера

Структура заголовка драйвера

смещение	тип поля	содержание поля
0	DD двойное слово	-1 (0FFFFFFFFh) При загрузке подставляется адрес следующего драйвера. Если драйвер последний в цепочке, то вместо смещения останется 0FFFFh.
4	DW	Атрибуты драйвера (подробно поле будет рассмотрено ниже)
6	DW	Смещение до начала ПРОЦЕДУРЫ СТРАТЕГИИ
8	DW	Смещение до начала ПРЕРЫВАНИЯ
0Ah	DB восемь байт	Для символьного драйвера - имя, а для блочного драйвера, первый байт этого поля - количество обслуживаемых этим драйвером устройств.

Поле атрибутов драйвера служит для идентификации типа драйвера (символьный или блочный), а также для информации о возможностях драйвера. Структура поля атрибутов драйвера приведена отдельно для символьных и отдельно для блочных драйверов.

Структура поля атрибутов символьного драйвера

Бит	Значение	Смысл
0	1	Устройство является стандартным устройством ввода (клавиатура).
1	1	Устройство является стандартным устройством вывода (дисплей).
2	1	Устройство является устройством типа NUL (отладочный драйвер).
3	1	Системные часы.
4-5	-	Резерв (должно быть ноль).
6	1	Устройство поддерживает функции DOS 3.2 (то есть, поддерживаются подфункции 0Dh - 0Fh функции DOSa 44h).
7-10	-	Резерв (должен быть ноль).
11	1	Устройство понимает команды OPEN/CLOSE.
12	-	Резерв (должен быть ноль).
13	1	Устройство поддерживает вывод OUTPUT UNTIL BUSY.
14	1	Драйвер поддерживает управление устройством с помощью контрольной строки управления вводом/выводом (IOCTL).
15	1	Признак символьного драйвера.

Структура поля атрибутов блочного драйвера

Бит	Значение	Смысл
0	-	Резерв (должен быть ноль).
1	1	Устройство поддерживает 32-х битный размер адреса сектора (начиная с DOSa 4.0).
2-5	-	Резерв (должен быть ноль).
6	1	Устройство поддерживает функции DOS 3.2.
7-10	-	Резерв (должен быть ноль).
11	1	Устройство понимает команды OPEN/CLOSE.
12	-	Резерв (должен быть ноль).
13	1	Устройство определяет тип носителя, проверяя байт идентификатора FAT.
14	1	Драйвер поддерживает управление устройством с помощью контрольной строки управления вводом/выводом (IOCTL).
15	0	Блочный драйвер

Процедура стратегии

Через процедуру стратегии, в паре регистров ES:BX, загружаемому драйверу передается адрес запроса драйвера (DRIVER REQUEST), в котором содержится команда, предназначенная для исполнения драйвером, и необходимые для этой команды данные. В то же поле заносится код возврата, вырабатываемый драйвером и данные передаваемые драйвером системе. Единственное назначение процедуры стратегии - сохранить адрес запроса драйвера (DRIVER REQUEST), чтобы впоследствии с этим запросом смог работать сам драйвером. В связи с этим, как правило, процедура стратегии состоит из команд сохранения регистров ES и BX и команды возврата в систему. Процедура стратегии должна быть оформлена как процедура типа FAR, так как она вызывается системой с помощью команды CALL FAR. Ниже представлен типичный пример процедуры стратегии:

```
STRATEGY    PROC    FAR
            MOV     CS:KEEP_BX, BX      ; сохраняем регистр BX
            MOV     CS:KEEP_ES, ES      ; сохраняем регистр ES
            RET                         ; возвращаемся в систему
KEEP_BX     DW      ?
KEEP_ES     DW      ?
STRATEGY    ENDP
```

Поскольку и процедура стратегии и процедура прерывания вызываются командой CALL FAR, причем система не принимает никаких мер, по сохранению и восстановлению регистров процессора, возникает необходимость особое внимание уделять корректному использованию, сохранению и восстановлению всех регистров. Так в примере процедуры стратегии, приведенном выше, для ячеек памяти явно указаны сегментные регистры, относительно которых эти переменные адресуются. Это сделано потому, что в момент, когда управление получает процедура стратегии, значение всех остальных сегментных регистров не определено (если адресация будет производится, например, относительно регистра DS, результат работы процедуры будет непредсказуемым). Кроме того, в этой процедуре мы не портим содержимое каких-либо регистров, иначе нам пришлось бы позаботиться о сохранении всех регистров, которыми мы пользуемся. Это необходимо будет сделать в процедуре прерывания.

Процедура прерывания

Процедура прерывания должна анализировать запрос драйвера, получать через него предназначенные ей данные, выполнять предусмотренные в ней команды, выдавать через это поле данные для системы и код возврата. В связи с этим запрос драйвера имеет жесткую структуру для каждой команды драйвера. Начинается каждый запрос драйвера с тринадцатибайтового поля заголовка запроса драйвера (REQUEST HEADER). Это поле одинаково для всех команд драйвера. Структура заголовка запроса драйвера:

смещение	тип поля	содержание поля
0	DB байт	Длина запроса драйвера в байтах.
1	DB байт	Номер устройства из числа управляемых драйвером, которому адресован этот запрос (только для блочных драйверов).
2	DB байт	Команда драйвера. Список возможных команд приведен ниже.
3	DW слово	Поле статуса драйвера. Заполняется драйвером. Структура этого поля также приведена ниже.
5	DB восемь байт	Резерв.

Команды драйвера

код команды	команда
0h 0	Команда начальной инициализации драйвера (INIT). Должна выполняться всеми драйверами.
1h 1	Команда проверки носителя (MEDIA CHECK). Только для блочных драйверов.
2h 2	Команда "построить блок параметров BIOSa" (BUILD BPB). Только для блочных драйверов.
3h 3	Команда "ввести строку управления вводом/выводом" (IOCTL INPUT).
4h 4	Ввод с устройства (INPUT).
5h 5	Неразрушающий ввод без ожидания (NON-DESTRUCTIVE READ, NO WAIT). Только для символьных драйверов.
6h 6	Ввод статуса (INPUT STATUS). Только для символьных драйверов.
7h 7	Очистить ввод (INPUT FLUSH). Только для символьных драйверов.
8h 8	Вывод на устройство (OUTPUT).
9h 9	Вывод с проверкой (OUTPUT WITH VERIFY).
0Ah 10	Вывод статуса (OUTPUT STATUS). Только для символьных драйверов.
0Bh 11	Очистить вывод (OUTPUT FLUSH). Только для символьных драйверов.
0Ch 12	Команда "вывести строку управления вводом/выводом" (IOCTL OUTPUT).

0Dh 13	Выполнить команду "открыть устройство" (OPEN DEVICE).
0Eh 14	Выполнить команду "закрыть устройство" (CLOSE DEVICE).
0Fh 15	Команда "сменить носитель" (REMOVABLE MEDIA).
10h 16	Вывод "пока занят" (OUTPUT UNTIL BUSY).
13h 19	Устройство поддерживает команду GENERIC IOCTL REQUEST (Подфункция 0Ch функции DOSa 44h).
17h 23	Получить логическое устройство (GET LOGICAL DEVICE).
18h 24	Установить логическое устройство (SET LOGICAL DEVICE).

Поле статуса

биты	смысл поля
0-7	Код ошибки. Поле работает только тогда, когда бит 15 установлен в единицу.
8	Бит устанавливается в единицу, когда драйвер кончил работу.
9	Бит устанавливается в единицу, когда драйвер занят.
10-14	Резерв (должен быть ноль).
15	Устанавливается в единицу, если при работе драйвера произошла ошибка.

Коды ошибок драйвера

код	вид ошибки
0	Попытка записи на защищенный от записи носитель.
1	Неизвестное устройство.
2	Устройство не готово.
3	Неизвестная команда.
4	Ошибка в контрольной сумме.
5	Плохая структура запроса драйвера (DRIVER REQUEST).
6	Ошибка поиска.
7	Неизвестный носитель.
8	Сектор не найден.
9	В принтере нет бумаги.

Структура запроса драйвера зависит от того, какая команда передана драйверу. Ознакомиться со структурой запроса для разных команд можно в фирменных руководствах [6,7]. Единственной командой, которую должен выполнять любой драйвер, является команда начальной инициализации драйвера (INIT). В примере 11 приведен текст простейшего драйвера, не выполняющего никаких действий, кроме начальной инициализации. Для запуска этого драйвера нужно его преобразовать к типу BIN, используя для этого программу EXE2BIN. Если имя драйвера PHANTOM и он помещен на диске C, то соответствующая строчка в файле конфигурации системы CONFIG.SYS должна выглядеть следующим образом:

DEVICE=C:\PHANTOM.BIN

Кроме того, используя программу, приведенную в примере 12, возможно, с помощью любого отладчика, проследить, как выполняется для этого драйвера команда INIT.

Пример 11. Простейший драйвер

```
CODE_SEG    SEGMENT
PHANTOM     PROC        FAR
            ASSUME      CS:CODE_SEG

;       заголовок драйвера
            DD          -1              ; при инициализации драйвера
                                        ; подставляется либо адрес
                                        ; следующего драйвера, либо
                                        ; признак последнего драйвера
ATTRIBUTE   DW          8000h           ; поле атрибутов
            DW          STRATEGY        ; смещение до процедуры стратегии
            DW          INTERRUPT       ; смещение до процедуры прерывания
            DB          'PHANTOM '      ; имя драйвера

;       процедура стратегии (она не обязательно следует сразу за заголовком драйвера, часто между
;       заголовком и процедурой стратегии располагают область данных)
STRATEGY    PROC        FAR             ; обязательно типа FAR
            MOV         CS:KEEP_BX, BX      ; сохраняем адрес запроса, который передается
            MOV         CS:KEEP_ES, ES      ; через пару регистров ES:BX
            RETF                            ; возвращаем управление системе
REQUEST     LABEL       DWORD
KEEP_BX     DW          ?
KEEP_ES     DW          ?
STRATEGY    ENDP
;       процедура прерывания (между процедурой прерывания и процедурой стратегии также можно
;       располагать данные)
INTERRUPT   PROC        FAR             ; обязательно типа FAR
;       сохраняем все используемые в этой процедуре регистры
            PUSH        DS
            PUSH        BX
            PUSH        SI

            LDS         SI, REQUEST     ; загружаем в пару регистров DS:SI указатель на
                                        ; запрос драйвера
            XOR         BX, BX          ; очищаем регистр BX
            MOV         BL, DS:[SI+2]   ; в регистр BL заносим команду
            SHL         BX, 1           ; удваиваем содержимое BX (это необходимо
                                        ; для использования таблицы переходов)
            JMP         CS:[TABLE+BX]   ; переходим по адресу из таблицы переходов
```

```
;       таблица выбора функций
TABLE           DW              INIT
                DW              CHECK_MEDIA
                DW              BUILD_BPB
                DW              IOCTL_INPUT
                DW              INPUT
                DW              READ_NO_WAIT
                DW              INPUT_STATUS
                DW              INPUT_FLUSH
                DW              OUTPUT
                DW              VERIFY_OUTPUT
                DW              OUTPUT_STATUS
                DW              OUTPUT_FLUSH
                DW              IOCTL_OUTPUT
                DW              DEVICE_OPEN
                DW              DEVICE_CLOSE
                DW              REMOVE_MEDIA
                DW              O_U_B
                DW              8 DUP(EXIT)

;       выполняем функцию инициализации драйвера
INIT:
;       в пакете запроса драйвера, в поле со смещением 0Eh помещается адрес точки, до которой код
;       драйвера остается резидентным (OFFSET)
                MOV             WORD PTR DS:[SI+0Eh], OFFSET END_OF_DRIVER
;       в пакете запроса драйвера, в поле со смещением 10h помещается сегмент резидентного кода
;       драйвера
                MOV             DS:[SI+10h], CS
                JMP             QUIT                    ; переход на выход из драйвера
;       для всех остальных функций вырабатывается код возврата "неизвестная команда"
CHECK_MEDIA:
BUILD_BPB:
IOCTL_INPUT:
INPUT:
READ_NO_WAIT:
INPUT_STATUS:
INPUT_FLUSH:
OUTPUT:
VERIFY_OUTPUT:
OUTPUT_STATUS:
OUTPUT_FLUSH:
IOCTL_OUTPUT:
DEVICE_OPEN:
DEVICE_CLOSE:
REMOVE_MEDIA:
O_U_B:
EXIT:
;       устанавливаем бит 15 - ошибка, код ошибки 3 - неизвестная
;       команда
                OR      WORD PTR DS:[SI+3], 8003h
```



```
QUIT:                                           ; выход из драйвера
;       устанавливаем бит 8 - драйвер окончил работу
                OR              WORD PTR DS:[SI+3], 0100h
;       восстанавливаем использованные регистры
                POP             SI
                POP             BX
                POP             DS
                RETF                            ; возвращаем управление DOS
END_OF_DRIVER:                                  ; адрес до которого код драй-
                                                ; вера остается резидентным
                                                ; при инициализации
INTERRUPT       ENDP
PHANTOM         ENDP
CODE_SEG        ENDS
                END
```

Особой проблемой является проблема отладки драйвера. Это связано с тем, что драйвер не является программой, которую можно было бы запустить внутри отладчика. Единственным способом, проверяющим работоспособность драйвера является включение его в файл CONFIG.SYS, после чего приходится ожидать результатов работы. Так, если неправильно выполняется команда инициализации (INIT), единственным признаком этого может оказаться "зависание" системы, что в данных условиях влечет за собой необходимость перезагружаться с другого носителя (с другим файлом CONFIG.SYS). В связи с этой проблемой обычно дают рекомендации такого типа, как написать сначала программу типа COM, отладить ее, а затем переделать в драйвер. Такой подход позволяет отладить ту часть программы, которая отвечает за связь с устройством или управление устройством, однако не позволяет отладить ту часть драйвера, которая отвечала бы за связь с операционной системой. В следующем примере (пример 12) приведена программа, позволяющая с помощью обычных отладчиков, отлаживать процесс инициализации драйверов. Сначала в буфер загружается текст драйвера, затем формируется запрос на выполнение команды INIT, затем передается управление процедурам стратегии и прерывания. При работе в отладчике нужно сразу же запустить программу с точкой останова CS:1000h, и начиная с этой точки трассировать исполнение процедур стратегии (первый вызов типа CALL FAR в программе) и прерывания (второй вызов типа CALL FAR в программе).

Естественно, поскольку эта книга является пособием, этот пример упрощен автором настолько, насколько это было возможным.

Пример 12. Программа, облегчающая отладку драйвера

```
DRIVER_SEG  SEGMENT
DRV_BUFFER  DW          8000h DUP(?)        ; буфер для отлаживаемого
                                            ; драйвера (размер 64K)
DRIVER_SEG  ENDS

STACK_SEG   SEGMENT     STACK
TOP_STACK   DW          800h DUP(?)         ; стек (размер стека четыре килобайта)
STACK_SEG   ENDS

DATA_SEG    SEGMENT                         ; сегмент данных
;           ********************************************************
REQUEST     LABEL       BYTE                ; пакет запроса команды инициализации (INIT)
LEN_REQ     DB          17h                 ; длина пакета запроса для команды INIT
UNIT_NUM    DB          0                   ; номер устройства (для блочных драйверов)
CMD_CODE    DB          0                   ; код команды INIT
STATUS      DW          ?                   ; статус драйвера (заполняется драйвером)
RESERV      DB          8 DUP(0)            ; резервное поле
NUM_OF_UN   DB          ?                   ; число устройств обслуживаемых драйвером
                                            ; Заполняется драйвером (Только для блочных
                                            ; устройств)
END_ADDR    DD          ?                   ; адрес конца резидентного кода
STRING_ADDR     DD      ?                   ; адрес области памяти, где содержится строка
                                            ; из CONFIG.SYS для этого драйвера. Блочный
                                            ; драйвер должен возвращать в этом поле адрес
                                            ; блока BPB этого устройства
DRIVE_NUMBER    DB      ?                   ; номер устройства обслуживаемого этим
                                            ; драйвером (0=A, 1=B и т.д.)
;           ********************************************************
DRIVER_NAME     DB      40h DUP(0)          ; буфер для имени драйвера (считывается из
                                            ; командной строки, через соответствующее
                                            ; поле блока PSP)
CONFIG_STRING   DB      40h DUP(0)          ; строка, адрес которой указывается в перемен-
                                            ; ной STRING_ADDR. Также считывается из
                                            ; командной строки нашей программы
HANDLE          DW      ?                   ; HANDLE файла, содержащего драйвер

;           ********************************************************
;           сообщения выдаваемые программой
MESSAGE_1   DB          0Dh, 0Ah, 7
            DB          9, 'ПУСТАЯ КОМАНДНАЯ СТРОКА!'
            DB          0Dh, 0Ah, '$'

MESSAGE_2   DB          0Dh, 0Ah, 7
            DB          9, 'НЕВЕРНОЕ ИМЯ ДРАЙВЕРА!'
            DB          0Dh, 0Ah, '$'
```



```
MESSAGE_3   DB          0Dh, 0Ah, 7
            DB          9, 'ОШИБКА ЧТЕНИЯ!'
            DB          0Dh, 0Ah, '$'
;           ********************************************************
;           здесь формируется адрес процедуры стратегии
STRATEGY_CALL   LABEL DWORD
STRATEGY        DW  ?
STRATEGY_SEG    DW  ?
;           здесь формируется адрес процедуры прерывания
INTERRUPT_CALL  LABEL DWORD
INTERRUPT       DW  ?
INTERRUPT_SEG   DW  ?
DATA_SEG    ENDS

CODE_SEG    SEGMENT
;           информация транслятору, о назначении сегментных регистров
            ASSUME      CS:CODE_SEG,DS:DRIVER_SEG,ES:DATA_SEG,SS:STACK_SEG
BEGIN:
;           проверяем наличие информации в командной строке
            XOR         CX, CX              ; очищаем регистр CX
            MOV         CL, DS:[80h]        ; заносим в регистр CL длину командной строки
            JCXZ        WRITE_MSG1          ; если командная строка отсутствует, то переход
                                            ; на вывод сообщения 1
            JMP         READ_DRV_NAME       ; переход на чтение имени загружаемого
                                            ; драйвера
WRITE_MSG1:
;           для вывода сообщения инициализируем регистр DS
            MOV         AX, DATA_SEG
            MOV         DS, AX
            LEA         DX, MESSAGE_1       ; заносим адрес сообщения в регистр DX
            MOV         AH, 9               ; заносим номер функции в
                                            ; регистр AX
            INT         21h                 ; вызов диспетчера DOSa
            JMP         QUIT                ; на выход из программы
READ_DRV_NAME:
;           здесь мы считываем имя драйвера и строку CONFIG.SYS для него
            DEC         CX                  ; пропускаем пробел, находящийся перед
                                            ; именем драйвера
            LEA         DI, DRIVER_NAME     ; заносим адрес строки для имени драйвера в
                                            ; регистр DI
            MOV         SI, 82h             ; в регистр SI заносим адрес начала имени
                                            ; драйвера (для упрощения текста программы
                                            ; предположим, что перед именем нет лишних
                                            ; пробелов)
            CLD                             ; устанавливаем направление работы строчных
                                            ; команд
            MOV         AX, DATA_SEG        ; инициализируем регистр
            MOV         ES, AX              ; ES
;           начало цикла чтения
READ_NAME:
            LODSB                           ; считываем байт из строки по адресу DS:SI в AL
```

```
                CMP     AL, 20h                 ; проверяем, не пробел ли это
                JZ      QUIT_LOOP               ; если пробел - выход из цикла
                STOSB                           ; записываем символ в строку имени драйвера
                LOOP    READ_NAME               ; переход к началу цикла
                MOV     BYTE PTR ES:[DI], 0     ; создаем строку типа ASCIIZ
                                                ; (оканчивающуюся 0)
                JMP     LOAD_DRV                ; если отсутствует строка для
                                                ; CONFIG_STRING, то сразу переходим на загрузку
                                                ; драйвера
QUIT_LOOP:
;       теперь цикл считывания строки для CONFIG_STRING

                LEA     DI, CONFIG_STRING       ; в регистр DI адрес строки CONFIG_STRING
        REP     MOVSB
LOAD_DRV:                                       ; считываем драйвер
                MOV     AH, 3Dh                 ; вызываем функцию DOSa 3Dh - открыть файл
                MOV     AL, 0                   ; файл открыть только для чтения
                MOV     DX, DATA_SEG            ; пара регистров DS:DX должна
                MOV     DS, DX                  ; содержать ASCIIZ строку - имя файла
                LEA     DX, DRIVER_NAME         ; адрес имени файла заносим в DX
                INT     21h                     ; делаем попытку открыть файл, содержащий
                                                ; отлаживаемый драйвер
                JNC     READ_DRIVER             ; если удалось открыть файл, то переходим на
                                                ; его чтение
;       если не удалось открыть файл, то сообщение об ошибке и выход из программы
                LEA     DX, MESSAGE_2           ; помещаем адрес сообщения в регистр DX
                MOV     AH, 09                  ; девятая функция - вывод строки на экран
                INT     21h                     ; выводим сообщение об ошибке на экран
                JMP     QUIT                    ; на выход
READ_DRIVER:
                MOV     HANDLE, AX              ; сохраняем HANDLE файла в ячейке памяти
                MOV     BX, AX                  ; посылаем HANDLE файла в регистр BX
                                                ; (требуется для выполнения функции чтения)
                MOV     AH, 3Fh                 ; заносим номер функции в регистр AH
;       занесем в пару регистров DS:DX адрес буфера, куда мы считываем файл
                MOV     DX, DRIVER_SEG
                MOV     DS, DX
                LEA     DX, DRV_BUFFER
;       в регистр CX помещаем количество считываемых байт, причем, если окажется, что размер
;       файла меньше чем CX, то в регистре AX всегда можно посмотреть истинное количество
;       прочитанных байт
                MOV     CX, 0FFFFh
                INT     21h                     ; делаем попытку прочитать файл
                JNC     FORM_REQUEST            ; если драйвер прочитан без ошибок,
                                                ; переходим на формирование запроса
;       в случае ошибки чтения выводим сообщение и переходим на конец программы
                MOV     DX, DATA_SEG            ; помещаем адрес сообщения
                MOV     DS, DX                  ; в пару регистров
                LEA     DX, MESSAGE_3           ; DS:DX
                MOV     AH, 09                  ; девятая функция - вывод
                INT     21h                     ; строчки на экран
```



```
                JMP     QUIT                    ; переход на конец программы
;       формирование запроса драйвера
FORM_REQUEST:
                MOV     BX, HANDLE              ; закрываем файл, содержащий
                MOV     AH, 3Eh                 ; отлаживаемый драйвер
                INT     21h                     ; (он файл нам больше не нужен)
;       заносим в поле STRING_ADDR адрес строки из "CONFIG.SYS"
;       (в нашей программе - адрес строки CONFIG_STRING)
                MOV     WORD PTR STRING_ADDR, OFFSET CONFIG_STRING
                MOV     WORD PTR STRING_ADDR[2], ES
;       для вызова процедур стратегии и прерывания формируем адреса этих процедур
                MOV     STRATEGY_SEG, DS        ; сегмент кода загружаемого
                MOV     INTERRUPT_SEG, DS       ; драйвера сейчас равен DS
                MOV     AX, DS:6                ; берем из заголовка драйвера
                MOV     STRATEGY, AX            ; адреса процедур стратегии и
                MOV     AX, DS:8                ; прерывания и заносим их в
                MOV     INTERRUPT, AX           ; смещение для вызова процедур
;       помещаем в регистр BX адрес запроса драйвера
                LEA     BX, REQUEST
                JMP     START_DRIVER
;       для удобства работы с отладчиком устанавливаем адрес 1000h
                ORG     1000h
START_DRIVER:
                CALL    STRATEGY_CALL           ; вызвали процедуру стратегии
                CALL    INTERRUPT_CALL          ; вызвали процедуру прерывания

;       выход из программы
QUIT:
                MOV     AH, 4Ch                 ; функция DOSa 4Ch - выход из программы
                MOV     AL, 0                   ; код возврата - 0
                INT     21h                     ; выполняем функцию
CODE_SEG        ENDS
                END     BEGIN
```

Для использования этой программы, ее надо запустить под управлением какого-либо отладчика, следующим образом:

Имя_отладчика Имя_программы Имя_драйвера параметры_драйвера;

Так например, если использовать эту программу для отладки драйвера, приведенного в примере 11, нужно написать следующую строчку (будем считать, что драйвер называется PHANTOM.BIN, программа называется LOADER.EXE, а в качестве отладчика используется отладчик AFD_PRO.EXE):

```
AFD_PRO LOADER.EXE PHANTOM.BIN
```

Внутри отладчика программу надо запустить до адреса 1000h (для AFD_PRO используется команда g 0,1000). Две команды CALL FAR вызывают соответственно программы STRATEGY и INTERRUPT. Эта программа, единственная в книге программа типа EXE.

Заключение

К сожалению, автор был ограничен объёмом книги и, поэтому, не сумел изложить все вопросы, представляющие интерес для человека, желающего использовать компьютер не только как игрушку, но и как серьезный инструмент. Однако, автор надеется, что книга будет представлять интерес и в таком виде. Примеры также подобраны, по возможности, так, чтобы от них была какая то практическая польза, хотя, безусловно, они были максимально упрощены.

Список использованной литературы:

1. **Richard Wilton.** Programmer's guide to PC & PS/2 video systems. Maximum Video. Performance from the EGA, VGA, HGC and MCGA. Microsoft Press, 1987, Washington.
2. **Peter Abel.** IBM PC Assembler Language and Programming. Prentice-Hall International, Inc. 1987, London.
3. **Р. Джордейн.** Справочник программиста персональных компьютеров типа IBM PC, XT и AT. "Финансы и статистика", 1991, Москва.
4. **П. Нортон.** Персональный компьютер фирмы IBM и операционная система MS DOS. "Радио и связь", 1991, Москва.
5. **Л. Дао.** Программирование микропроцессора 8088. "МИР", 1988, Москва.
6. Microsoft. MS-DOS. Programmer's Reference. Version 4.x. Microsoft Press, 1988, Washington.
7. MS-DOS Programmer's Reference. Including Version 5. Microsoft Press, 1991, Washington.
8. Enchanced Graphics Adapter. Technical Manual. Gemini Technology Inc., 1988.
9. **Ю-Чжен Лю, Г. Гибсон.** Микропроцессоры семейства 8086/8088. Архитектура, программирование и проектирование микрокомпьютерных систем. "Радио и связь", 1987, Москва.

Приложение. Список команд Ассемблера, использованных в книге

ADC **OP1, OP2** - Команда сложить с учетом переноса. К содержимому OP1 добавляется содержимое OP2 и флаг переноса CF. Результат записывается в OP1. Команда действует на все арифметические флаги (CF, ZF, PF, AF, SF, OF).
В качестве операндов могут быть использованы:

1. Два регистра.
2. Регистр и ячейка памяти.
3. Регистр и непосредственный операнд.
4. Ячейка памяти и непосредственный операнд.

Имеются следующие ограничения на типы операндов:

1. Операнды не могут быть одновременно ячейками памяти.
2. Операндами не могут быть сегментные регистры.
3. OP1 не может быть непосредственным значением.
4. Операнды должны иметь одинаковый размер.

ADD **OP1, OP2** - Команда сложить. К содержимому OP1 добавляется содержимое OP2. Результат записывается в OP1. Команда действует на все арифметические флаги (CF, ZF, PF, AF, SF, OF).
В качестве операндов могут быть использованы:

1. Два регистра.
2. Регистр и ячейка памяти.
3. Регистр и непосредственный операнд.
4. Ячейка памяти и непосредственный операнд.

Имеются следующие ограничения на типы операндов:

1. Операнды не могут быть одновременно ячейками памяти.
2. Операндами не могут быть сегментные регистры.
3. OP1 не может быть непосредственным значением.
4. Операнды должны иметь одинаковый размер.

AND **OP1, OP2** - Логическое и. Побитовое логическое и. Результат записывается в OP1. Команда действует на флаги: SF; ZF; Pf. Устанавливает в 0 флаги CF и OF. Использование операндов такое же, как и в команде ADD. Таблица истинности для одного бита:

OP1	OP2	результат
0	0	0
0	1	0
1	0	0
1	1	1

CALL **ADDRESS** - Команда вызова подпрограммы. Вызывается подпрограмма, расположенная по адресу ADDRESS. Команда CALL бывает двух типов:

1. Команда типа NEAR - в этом случае используется двухбайтовый адрес. Адрес возврата - одно слово - заносится в верхушку стека.
2. Команда типа FAR - в этом случае используется четырехбайтовый адрес. Адрес возврата - два слова - заносится в верхушку стека.

В этой команде не передаются параметры, поэтому программист должен сам выбирать способ передачи параметров подпрограмме.

CLC - Сбросить флаг переноса - CF.

CLD - Сбросить флаг направления - DF.

CLI - Сбросить флаг прерывания - IF (запретить внешние, маскируемые прерывания).

CMC - Инвертировать флаг переноса - CF.

CMP **OP1, OP2** - Сравнить. Выставляет флажки так же, как и команда вычитания, но операнды не меняются. Возможные операнды такие же, как и в команде ADD.

CMPS - Сравнить цепочки. Имеет две формы:
Сравнить цепочку байт - CMPSB;
Сравнить цепочку слов - CMPSW.
Сравнивает ячейку памяти расположенную по адресу DS:SI с ячейкой памяти по адресу ES:DI. Если флаг DF=0, то после сравнения содержимое регистров DI и SI увеличивается (на единицу для CMPSB, или на два для CMPSW). Если флаг направления DF=1, то содержимое регистров DI и SI уменьшается.

DEC **OP1** - Уменьшить содержимое операнда OP1 на единицу. Действует на все арифметические флаги, кроме флага CF. Операндами могут быть регистры или ячейка памяти.

IN **REG_ A, PORT** - Ввести из порта в регистр AL или AX. Порт - это или непосредственное значение - один байт, или регистр DX.

INC **OP1** - Увеличить содержимое операнда OP1 на единицу. Действует на все арифметические флаги, кроме CF. Операндами могут быть регистры или ячейка памяти.

INT **BYTE** - Вызов программного прерывания. Вызывает подпрограмму обработки прерывания, адрес которой находится в таблице векторов прерываний. Сбрасывает флаги IF и TF. Помимо полного адреса возврата (CS и IP), сохраняет в стеке слово состояния процесса - PSW.

IRET - Возвращение из подпрограммы обработки прерывания. Восстанавливает PSW, сохраненное в стеке.

JMP **ADDRESS** - Команда безусловного перехода. Переход на адрес ADDRESS. Эта команда бывает трех типов:

1. Команда типа SHORT - адрес перехода - один байт (короткий переход). Переход осуществляется в пределах -128...+127 байтов от команды, следующей за командой условного перехода.
2. Команда типа NEAR - адрес перехода - одно слово (два байта). Осуществляет переход в любую точку сегмента. Это - команда внутрисегментного перехода.
3. Команда типа FAR - адрес перехода - два слова. Команда дальнего, или межсегментного, перехода.

J(CC) **ADDRESS** - Группа команд условного перехода (CC - Condition Code - код возврата). Адрес в этой команде всегда один байт, то есть все команды условного перехода относятся к командам короткого перехода. Проверяется состояние флагов PSW. Если состояние флагов отвечает определенным условиям, следует переход по адресу ADDRESS. В противном случае, выполняется следующая по порядку команда. Иногда, для одних и тех же команд условного перехода, используется различная мнемоника. Программист выбирает для программы ту, которая в данном случае является более подходящей. Ниже перечислены команды условного перехода и проверяемые ими флаги.

	мнемоника и формат	альтернативная мнемоника	флаги
ПЕРЕЙТИ,			
если 0	JZ	JE	ZF=1
если не 0	JNZ	JNE	ZF=0

ПЕРЕЙТИ,			
если знак есть	JS	-	SF=1
если знака нет	JNS	-	SF=0
если есть переполнение	JO	-	OF=1
если нет переполнения	JNO	-	OF=0
если четный паритет	JP	JPE	PF=1
если нечетный паритет	JNP	JPO	PF=0
если меньше (без знака)	JB	JNAE, JC	CF=1
если не меньше (без знака)	JNB	JAE, JNC	CF=0
если меньше или равно (без знака)	JBE	JNA	CF & ZF=1
если больше (без знака)	JA	JNBE	CF & ZF=0
если меньше (со знаком)	JL	JNGE	SF ! OF=1
если больше или равно (со знаком)	JGE	JNL	SF ! OF=0
если меньше или равно (со знаком)	JLE	JNG	(SF ! OF) & ZF=1
если больше (со знаком)	JG	JNLE	(SF ! OF) & ZF=0

JCXZ ADDRESS - Команда перехода по адресу ADDRESS, в случае, если содержимое регистра CX равно 0. Адрес однобайтовый, следовательно, это команда короткого перехода.

LDS REG, MEM - Загрузка пары регистров DS:REG содержимым четырехбайтовой ячейки памяти MEM.

LEA REG, SRC - Загрузка эффективного адреса источника SRC в регистр REG.

LODSB - Загрузить цепочку байт. Загрузить байт в регистр AL из ячейки памяти, по адресу DS:SI. Если флаг направления DF=0, то содержимое регистра SI при этом увеличивается на единицу. Если DF=1, то уменьшается на единицу.

LODSW - Загрузить цепочку слов. Загрузить слово в регистр AX из ячейки памяти, по адресу DS:SI. Если флаг направления DF=0, то содержимое регистра SI при этом увеличивается на два. Если DF=1, то уменьшается на два.

LOOP ADDRESS - Команда цикла. Относится к командам условного перехода. Счетчик цикла располагается в регистре CX. Сначала выполняется уменьшение CX на единицу, а затем, если CX не равен нулю, следует переход по адресу ADDRESS. Команда осуществляет короткий переход (адрес перехода один байт).

LOOPE ADDRESS - Команда цикла. Разновидность команды LOOP. Проверяется флаг ZF. Переход осуществляется, если CX не равен 0 и ZF=1.

LOOPNE ADDRESS - Команда цикла. Разновидность команды LOOP. Проверяется флаг ZF. Переход осуществляется, если CX не равен 0 и ZF=0.

MOV OP1, OP2 - Команда пересылки данных. Данные пересылаются из OP2 в OP1. Команда MOV, безусловно, наиболее часто используемая команда в реальных программах. Команда MOV не действует на флаги. В качестве операндов могут выступать:

1. Два регистра.
2. Регистр и ячейка памяти.
3. Регистр и непосредственный операнд.
4. Ячейка памяти и непосредственный операнд.

Имеются следующие ограничения на типы операндов:

1. Невозможна пересылка типа "ПАМЯТЬ - ПАМЯТЬ".
2. OP1 не может быть регистром кодового сегмента CS.
3. OP1 не может быть непосредственным значением.
4. Невозможна пересылка непосредственного значения в сегментный регистр.
5. Невозможна пересылка "СЕГМЕНТНЫЙ РЕГИСТР - СЕГМЕНТНЫЙ РЕГИСТР".
6. Естественно, невозможна пересылка в непосредственное значение.
7. OP1 и OP2 должны иметь одинаковый размер.

MOVSB - Пересылка цепочки байтов из области памяти DS:SI в область памяти ES:DI. Если флаг CF=0, то после пересылки происходит автоматическое увеличение содержимого регистров SI и DI на 1. Если CF=1, то после пересылки происходит автоматическое уменьшение содержимого регистров DS:SI и ES:DI на 1.

MOVSW - Пересылка цепочки слов из области памяти DS:SI в область памяти ES:DI. Если флаг CF=0, то после пересылки происходит автоматическое увеличение содержимого регистров SI и DI на 2. Если CF=1, то после пересылки происходит автоматическое уменьшение содержимого регистров DS:SI и ES:DI на 2.

MUL OP1 - Умножить содержимое регистра AL, или AX, на операнд OP1. Результат помещается в регистр AX (для умножения операндов размерностью один байт), или в пару регистров (DX:AX). Операндом OP1 может быть, или ячейка памяти, или регистр, кроме сегментного. Для микропроцессоров 80286 и старше, операнд может быть непосредственным значением.

NEG OP1 - Заносит в операнд OP1 значение 1-OP1 (изменить знак операнда). Операнд - регистр, или ячейка памяти. Не может быть сегментным регистром.

OR OP1, OP2 - Логическое или. Побитовое логическое или. Результат записывается в OP1. Использование операндов такое же, как и в команде ADD. Таблица истинности для одного бита:

OP1	OP2	результат
0	0	0
0	1	1
1	0	1
1	1	1

OUT PORT, REG_A - Вывести из регистра AL, или регистра AX, в порт PORT. В качестве порта вывода может выступать или непосредственное значение, или регистр DX. Непосредственное значение не превышает 255 (0FFh).

POP **OP1** - Извлечь слово из стека в операнд OP1. В качестве операнда OP1 могут выступать или регистр (кроме CS), или ячейка памяти.

POPF - Извлечь из стека PSW.

PUSH **OP1** - Записать слово в стек. OP1 может быть любым регистром, или ячейкой памяти. Для процессоров 80286 и старше, операнд OP1 может быть непосредственным значением.

PUSHF Записать в стек PSW.

REP - Префикс повторения для цепочечных команд. Уменьшает содержимое CX на единицу и, если CX не равен нулю, повторяет цепочечную команду.

REPE - Префикс повторения для цепочечных команд. "Повторять пока равно". Вариант префикса REP. Повторяет цепочечную операцию до тех пор, пока CX не равен нулю и, одновременно, флаг ZF=1. Для корректной работы этого префикса необходимо, перед выполнением команды, установить флаг ZF равным единице.

REPNE - Префикс повторения для цепочечных команд. "Повторять пока не равно". Вариант префикса REP. Повторяет цепочечную команду до тех пор, пока CX не равен нулю и, одновременно, флаг ZF=0.

RET - Возвращение из подпрограммы. Так как существуют подпрограммы типа FAR и типа NEAR, то, соответственно, различают и команды RET таких же типов. Команда RET типа FAR может быть явно определена для транслятора MASM 5.0 и старше в виде RETF. Команда RET типа NEAR может быть явно задана для транслятора MASM 5.0 и старше в виде RETN. Для других трансляторов тип команды RET определяется только из описателя процедуры PROC. В команде RETF в качестве адреса возврата используются два слова с верхушки стека, а в команде RETN, одно слово с верхушки стека.

SBB **OP1, OP2** - Вычесть содержимое OP2 из OP1, вычесть из OP1 флаг CF. То есть, вычитание, с учетом переноса. Влияет на все арифметические флаги (CF, ZF, PF, AF, SF, OF).
Использование операндов такое же, как и в команде ADD.

SCASB - Сканировать цепочку байт. Сравнивает содержимое ячейки памяти по адресу DS:SI с ячейкой памяти по адресу ES:DI. Если флаг направления DF=0, то содержимое регистров SI и DI увеличивается на единицу. Если же флаг DF=1, то содержимое регистров SI и DI уменьшается на единицу.

SCASW - Сканировать цепочку слов. Сравнивает содержимое ячейки памяти по адресу DS:SI с ячейкой памяти по адресу ES:DI. Если флаг направления DF=0, то содержимое регистров SI и DI увеличивается на два. Если же флаг DF=1, то содержимое регистров SI и DI уменьшается на два.

SHL **OP1, 1** - Сдвинуть содержимое операнда OP1 на 1 влево. Старший бит при этом теряется. Действует на флаги OF, SF, ZF, PF и CF.

SHL **OP1, CL** - Сдвинуть содержимое операнда OP1 на CL влево. Старшие CL бит при этом теряются. Действует на флаги OF, SF, ZF, PF и CF.

SHR **OP1, 1** - Сдвинуть содержимое операнда OP1 на 1 вправо. Старший бит при этом теряется. Действует на флаги OF, SF, ZF, PF и CF.

SHR **OP1, CL** - Сдвинуть содержимое операнда OP1 на CL вправо. Старшие CL бит при этом теряются. Действует на флаги OF, SF, ZF, PF и CF.

STC - Установить флаг переноса CF.

STD - Установить флаг направления DF.

STI - Установить флаг разрешения прерывания IF.

STOSB - Сохранить в памяти цепочку байт. Записывается в ячейку памяти по адресу ES:DI регистр AL. Если флаг направления DF=0, то содержимое регистра DI увеличивается на единицу. Если DF=1, то уменьшается на единицу.

STOSW - Сохранить в памяти цепочку слов. Записывается в ячейку памяти по адресу ES:DI регистр AL. Если флаг направления DF=0, то содержимое регистра DI увеличивается на два. Если DF=1, то уменьшается на два.

SUB **OP1, OP2** - Вычесть содержимое OP2 из OP1. Результат записать в OP1. Влияет на все арифметические флаги (CF, ZF, PF, AF, SF, OF).
Использование операндов такое же, как и в команде ADD.

TEST **OP1, OP2** - Проверить. Устанавливает флаги так же, как и команда AND (логическое и), однако не меняет содержимое операндов.

XCHG **OP1, OP2** - Обменять содержимое операндов. Ни один из операндов не может быть сегментным регистром. Ни один из операндов не может быть непосредственным значением. Остальные ограничения на операнды такие же, как и в команде MOV.

XLAT - Команда перекодировки. Операндов не имеет. Замещает содержимое регистра AL содержимым ячейки памяти, находящейся по адресу DS:[BX+AL], то есть регистр BX выступает в качестве базового, а регистр AL в качестве индексного.

XOR **OP1, OP2** - Исключающее или. Побитовое исключающее или. Результат записывается в OP1
Использование операндов такое же, как и в команде ADD. Таблица истинности для одного бита:

OP1	OP2	результат
0	0	0
0	1	1
1	0	1
1	1	0

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ НА АССЕМБЛЕРЕ ДЛЯ IBM-совместимых персональных компьютеров.

Богословский А.В.

Подписано в печать 30.5.92 Формат 60*90 1/16. Тираж 20 тыс.экз., 6 п.л. Заказ 56
Издатель: Малое индивидуальное предприятие Память, 115470, Москва, а/я 13.
Отпечатано