Резидентные программы

1. Что такое резидентная программа?

Обычно после завершения очередной выполняющейся программы DOS освобождает место в памяти, которое занимала программа, чтобы загрузить на это место новую. Однако есть способ оставить программу в памяти и после ее завершения. Следующая запускаемая программа при этом разместится в памяти после оставленной.

Остающиеся в памяти после завершения своей работы программы называются резидентными или TSR (Terminate and Stay Resident).

Для чего может понадобиться составление TSR-программ?

Существует несколько приложений, для которых подходят TSR-программы. Чаще всего TSR-программы используются для русификации импортных персональных компьютеров. Написано множество программ для загрузки русских шрифтов в память видеоадаптеров, для печати русских букв на принтере в графическом режиме, для русификации клавиатуры и т.п. Для всех этих программ характерно то, что они запускаются один раз при загрузке компьютера - их имена обычно включают в AUTOEXEC.BAT. Эти программы могут переключать на себя обработку прерываний, связанных с выводом на печать или с обращением к клавиатуре и/или выполнять разовые инициализирующие действия, такие как загрузка русских шрифтов в память видеоадаптера.

Другим примером использования TSR-программ могут служить программы резидентных калькуляторов, справочных баз данных типа Norton Guide или целых интегрированных систем наподобие Sidekick фирмы Borland. Такие программы тоже обычно запускаются через AUTOEXEC.BAT или при необходимости. Они перехватывают клавиатурные прерывания и отслеживают нажатие клавиш. Как только обнаруживается нажатие определенной заранее клавиши, поверх имеющегося на экране изображения выводится окно диалога резидентной программы.

Иногда TSR-программы используют для обслуживания нестандартной аппаратуры, особенно когда с этой аппаратурой должны работать по очереди несколько разных программ. В этом случае TSR-программа встраивает обработчик какого-либо прерывания, не занятого системой, и через этот обработчик все прикладные программы обращаются к нестандартной аппаратуре.

Аналогично работает резидентная часть системы управления базами данных ORACLE. Через прерывание, устанавливаемое при запуске ORACLE, прикладная программа, составленная на любом языке программирования, имеющем средство вызова прерывания, может пользоваться базой данных ORACLE.

Для обслуживания нестандартной аппаратуры больше подходят загружаемые драйверы устройств, о которых мы еще будем подробно говорить. Драйверы предоставляют намного более гибкие и богатые средства общения с устройствами, чем TSR-программы. Однако, если все, что вы собираетесь сделать, - это обработать несколько прерываний или обслужить какое-нибудь несложное устройство, то TSR-программы - это то, что вам нужно.

TSR-программы не могут свободно пользоваться прерываниями DOS из-за того, что программы DOS (и BIOS) не обладают свойством реентерабельности. Об этом мы будем говорить при обсуждении особенностей резидентных программ.

2. Как программе стать резидентной?

У вас есть две возможности оставить свою программу резидентной в памяти - использовать прерывание INT 27h или функцию 31h прерывания INT 21h.

Для использования прерывания 27h сегментный регистр CS должен указывать на PSP программы, а в регистре DX должно быть записано смещение последнего байта программы плюс один байт. Нетрудно заметить, что этот способ остаться резидентной больше всего подходит для программ в формате COM. Вы не сможете оставить резидентной программу длиннее 64 килобайт.

Другой, более удобный способ - использовать функцию 31h прерывания INT 21h. В регистре AL вы можете указать код завершения программы, регистр DX в этом случае должен содержать длину резидентной части программы в параграфах. Здесь уже нет ограничения 64 килобайта на длину программы. Использование этой функции - единственная возможность оставить резидентной программу длиннее 64 килобайт.

Но не стоит увлекаться длинными TSR-программами, так как обычно освободить память, занимаемую ставшей уже ненужной резидентной программой, можно только с помощью перезагрузки операционной системы.

Библиотека функций Quick C содержит специальную функцию для оставления программы резидентной в памяти. Эта функция использует прерывание INT 21h (функция 31h) и имеет имя _dos_keep(). Первый параметр функции - код завершения (то, что записывается в регистр AL), а второй - длина резидентной части программы в параграфах.

Нужно внимательно следить за размером оставляемой резидентной части программы, не забывать про области данных и буфера. Если вы укажите слишком малый объем памяти для TSR-программы, то прикладная программа, загруженная сразу за ней, может оказаться разрушенной.

3. Вызов резидентной программы

Возможны два способа вызова резидентной программы: либо прикладная программа выдает прерывание, обрабатываемое TSR-программой, либо сама TSR-программа отслеживает нажатие клавиш оператором компьютера и, в случае нажатия определенной клавиши (или комбинации клавиш), запускает диалоговую часть резидентной программы.

Первый способ используется прикладными программами, работающими с нестандартной аппаратурой, второй - диалоговыми резидентными программами.

Самый простой способ организовать связь оператора с TSR-программой - это использовать прерывание по нажатию клавиши печати содержимого экрана PrtScr. Но при этом вы теряете возможность распечатки содержимого экрана. Впрочем, иногда вам и нужно, чтобы вместо печати содержимое экрана записывалось, например, в файл. Или вы можете составить свою собственную программу печати содержимого экрана (используя графическую печать или какие-либо особенности принтеров).

Если вы используете прерывание 9, вырабатываемое при каждом нажатии на клавишу для определения момента запуска диалога, не следует забывать об организации цепочки прерываний, с тем чтобы после анализа (или перед анализом) передать управление стандартному обработчику 9-го прерывания.

Подробно о работе с клавиатурой и о 9-м прерывании будет рассказано во втором томе книги.

4. Особенности резидентных программ

TSR-программы имеют некоторые особенности, отличающие их от "нормальных" программ.

Им не разрешается использовать DOS-прерывания, когда вздумается. Это связано с тем, что DOS проектировалась как однозадачная операционная система, поэтому модули DOS не обладают свойством реентерабельности (повторной входимости). Что это означает на практике?

Допустим, Ваша программа записывает длинный файл на диск. Во время записи вы (возможно, случайно) нажали клавишу, активизирующую TSR-программу записи содержимого экрана в файл.

Теперь доступа к диску требуют две программы - прикладная, записывающая длинный файл, и Ваша TSR-программа. Запись файла из прикладной программы приостановится, далее произойдет запись копии экрана в файл, после чего возобновится запись файла из прикладной программы. Все было бы хорошо, если бы прикладная программа и TSR-программа не использовали одни и те же внутренние области данных DOS для работы с диском. При запуске TSR-программа безвозвратно испортит текущее состояние служебных областей данных, которые прикладная программа использовала при записи на диск.

И таких примеров можно привести много. BIOS также далеко не весь реентерабельный. TSR-программа может смело использовать разве лишь прерывание 16h для работы с клавиатурой, которое реентерабельно. Для вывода на экран лучше всего использовать непосредственную запись символов в видеопамять дисплейного адаптера.

Не стоит пользоваться многими функциями библиотеки Quick C, так как они могут вызывать прерывания DOS. Например, функция malloc() вызывает прерывание DOS для определения размера свободной памяти в системе.

Могут возникнуть трудности с использованием арифметических действий с числами в формате плавающей запятой, так как функция _dos_keep() при завершении программы восстанавливает прерывания, использовавшиеся для эмуляции арифметики с плавающей запятой и для работы с арифметическим сопроцессором.

Некоторые из перечисленных проблем (те, что связаны с использованием прерываний DOS) можно решить с помощью недокументированного прерывания INT 28h.

Это прерывание вызывается DOS при ожидании ввода с клавиатуры. В этот момент вы можете использовать любое прерывание DOS, кроме функций от 00h до 0Ch прерывания INT 21h. Утилита спулинга печати PRINT.COM использует это прерывание.

Можно рекомендовать следующий способ использования прерывания 28h. Обработчик прерывания 9 отслеживает нажатие клавиши, которая должна активизировать TSR-программу. Обнаружив эту клавишу (или комбинацию клавиш), обработчик прерывания 9 устанавливает флаг запроса на активизацию TSR-программы и завершает свою работу обычным способом.

Ваша TSR-программа должна создать свой обработчик прерывания 28h и сцепить его со стандартным. Каждый раз, когда DOS ожидает ввода с клавиатуры (в этот момент DOS не использует сама свои прерывания), вызывается прерывание 28h. Ваш обработчик проверяет флаг активизации, устанавливаемый обработчиком прерывания 9, и если флаг установлен, TSR-программа активизируется и может пользоваться услугами DOS, в частности, файловой системой.

Разумеется, после выполнения всех необходимых действий TSR-программа должна сбросить флаг активизации.

Можно также вместе с прерыванием 28 использовать аппаратное прерывание таймера с номером 8. В этом случае надо проверять не только флаг активизации, но и так называемый флаг критической секции DOS. Это байт, адрес которого возвращает недокументированная функция 34h прерывания DOS 21h в регистрах ES:BX. Если этот байт равен 0, то DOS не использует свои прерывания и наступил подходящий момент для активизации TSR-программы.

TSR-программа может вступить в конфликт с другими TSR-программами или прикладным обеспечением, если будет использовать занятые ими номера прерываний. Известен случай, когда драйвер клавиатуры и экрана (SDRIVER) вызывал зависания программных продуктов фирмы Microsoft из-за того, что эти продукты выдавали 16h прерывание с содержимым регистра AX равным 5500h, 5501h и т.д. Обработчик 16h-го прерывания программы SDRIVER при этом зацикливался из-за ошибки в программе (или точнее, из-за плохой реакции на такого рода номера функций прерывания 16h).

Очень важно не допустить случайного повторного запуска TSR-программы, так как повторное переназначение векторов прерываний почти наверняка приведет систему к краху.

Для проверки наличия TSR-программы в памяти обычно используют прерывание мультиплексора 2Fh, специально предназначенное для организации элементов мультизадачности в DOS. Это прерывание используется спулером печати PRINT.COM (об этом мы будем говорить при описании работы с принтером).

TSR-программа может переназначить это прерывание на себя и сделать так, чтобы оно "откликалось" на какой-либо код функции, зарезервированный для прикладных программ. Можно использовать коды C0h...FFh. При запуске TSR-программа вызывает прерывание 2Fh с выбранным кодом и проверяет ответ (передаваемый, например, в регистре AX). Если прерывание отвечает тем кодом, который задан в TSR-программе, это означает, что копия программы уже есть в памяти и повторная установка недопустима.

Приведенные ниже примеры проиллюстрируют все эти особенности.

5. Примеры резидентных программ

Приведем несколько примеров TSR-программ.

Первая программа перехватывает прерывание 9 (аппаратное прерывание клавиатуры). Запустив эту программу из приглашения DOS, вы сможете убедиться в том, что прерывание от клавиатуры возникает не только тогда, когда вы нажимаете на клавишу, но и когда ее отпускаете.

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

// Выключаем проверку стека и указателей
#pragma check_stack( off )
#pragma check_pointer( off )

// Макро для подачи звукового сигнала
#define BEEP() _asm { \
                      _asm xor  bx, bx    \
                      _asm mov  ax, 0E07h \
                      _asm int  10h       \
                    }

// Указатель на старую функцию обработки
//   9-го прерывания
void (_interrupt _far *oldkey)( void );

// Объявление новой функции обработки
//   9-го прерывания
```

```
void _interrupt _far newkey( void );

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

        unsigned size; // Размер резидентной части
                       // TSR-программы

        char _far *newstack; // Указатель на новый стек,
                             // который будет использовать
                             // TSR-программа

        char _far *tsrbottm; // Указатель на конец
                             // TSR-программы, используется
                             // для определения размера
                             // резидентной части

        // Записываем адрес стека TSR-программы
        _asm mov  WORD PTR newstack[0], sp
        _asm mov  WORD PTR newstack[2], ss

        FP_SEG(tsrbottm) = _psp; // Указатель конца
        FP_OFF(tsrbottm) = 0;    // программы устанавливаем
                                 // на начало PSP

        // Вычисляем размер программы в параграфах
        // Добавляем 1 параграф на случай
        // некратной параграфу длины
        size = ((newstack - tsrbottm) >> 4) + 1;

        // Встраиваем свой обработчик прерывания 9,
        // запоминаем старый вектор прерывания 9
        oldkey = _dos_getvect(0x9);
        _dos_setvect(0x9, newkey);

        // Завершаем программу и остаемся в памяти
        _dos_keep(0, size);
}

// Новый обработчик клавиатурного прерывания
void _interrupt _far newkey() {
        BEEP(); // Выдаем звуковой сигнал

        // Вызываем стандартный обработчик прерывания 9
        _chain_intr( oldkey );
}
```

Следующая программа GRAB демонстрирует использование функций DOS в TSR-программах. Она содержит все элементы, необходимые "безопасным" резидентным программам.

Программа предназначена для копирования содержимого видеобуфера в файл. Запись в файл активизируется при нажатии комбинации клавиш Ctrl+PrtSc. После каждой записи имя файла изменяется.

В самом начале своей работы программа проверяет наличие своей копии в памяти, так как повторное переназначение векторов прерываний приведет систему к краху. Некоторые тонкости, связанные с программированием клавиатуры и видеоадаптера, с получением адреса видеобуфера, используются в программе без объяснения. Вся необходимая информация будет приведена позже, в главах, посвященных программированию клавиатуры и видеоадаптеров.

Итак, текст программы:

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

// Выключаем проверку стека и указателей
#pragma check_stack( off )
#pragma check_pointer( off )

// Макро для подачи звукового сигнала
#define BEEP() _asm { \
                      _asm xor  bx, bx    \
                      _asm mov  ax, 0E07h \
                      _asm int  10h       \
                    }
// Указатели на старые обработчики прерываний
void (_interrupt _far *old8)(void);  // Таймер
void (_interrupt _far *old9)(void);  // Клавиатура
void (_interrupt _far *old28)(void); // Занятость DOS
void (_interrupt _far *old2f)(void); // Мультиплексор

// Новые обработчики прерываний
void _interrupt _far new8(void);
void _interrupt _far new9(void);
void _interrupt _far new28(void);
void _interrupt _far new2f(unsigned _es, unsigned _ds,
                           unsigned _di, unsigned _si,
                           unsigned _bp, unsigned _sp,
                           unsigned _bx, unsigned _dx,
                           unsigned _cx, unsigned _ax,
                           unsigned _ip, unsigned _cs,
                           unsigned flags);

int iniflag; // Флаг запроса на вывод экрана в файл
int outflag; // Флаг начала вывода в файл
int name_counter; // Номер текущего выводимого файла
char _far *crit;  // Адрес флага критической секции DOS

// =====================================
void main(void);
void main(void) {
         union REGS inregs, outregs;
         struct SREGS segregs;

         unsigned size; // Размер резидентной части
                        // TSR-программы

        // Вызываем прерывание мультиплексора с AX = FF00
        // Если программа GRAB уже запускалась, то новый
        // обработчик прерывания мультиплексора вернет
        // в регистре AX значение 00FF.
        // Таким способом мы избегаем повторного изменения
        // содержимого векторной таблицы прерываний.
        inregs.x.ax = 0xff00;
        int86(0x2f, &inregs, &outregs);

        if(outregs.x.ax == 0x00ff) {
                printf("\nПрограмма GRAB уже загружена\n");
                hello();
                exit(-1);
        }

        // Выдаем инструкцию по работе с программой GRAB
        hello();
```

```
        // Вычисляем размер программы в параграфах
        // Добавляем 1 параграф на случай
        // некратной параграфу длины
        size = (12000 >> 4) + 1;

        // Устанавливаем начальные значения флагов
        outflag=iniflag=0;

        // Сбрасываем счетчик файлов. Первый файл будет
        // иметь имя GRAB0.DOC. В дальнейшем этот счетчик
        // будет увеличивать свое значение на 1.
        name_counter=0;

        // Получаем указатель на флаг критической секции DOS.
        // Когда этот флаг равен 0, TSR-программа может
        // пользоваться функциями DOS
        inregs.h.ah = 0x34;
        intdosx( &inregs, &outregs, &segregs );
        crit=(char _far *)FP_MAKE(segregs.es,outregs.x.bx);

        // Устанавливаем собственные обработчики прерываний.
        old9 = _dos_getvect(0x9);
        _dos_setvect(0x9, new9);

        old8 = _dos_getvect(0x8);
        _dos_setvect(0x8, new8);

        old28 = _dos_getvect(0x28);
        _dos_setvect(0x28, new28);

        old2f = _dos_getvect(0x2f);
        _dos_setvect(0x2f, new2f);

        // Завершаем программу и остаемся в памяти
        _dos_keep(0, size);
}

// =====================================
// Новый обработчик прерывания мультиплексора.
// Используется для предохранения программы
// от повторного встраивания в систему как резидентной.

void _interrupt _far new2f(unsigned _es, unsigned _ds,
                           unsigned _di, unsigned _si,
                           unsigned _bp, unsigned _sp,
                           unsigned _bx, unsigned _dx,
                           unsigned _cx, unsigned _ax,
                           unsigned _ip, unsigned _cs,
                           unsigned flags)
{
                // Если прерывание вызвано с содержимым
                // регистра AX, равным FF00, возвращаем
                // в регистре AX значение 00FF,
                // в противном случае передаем управление
                // старому обработчику прерывания

                if(_ax != 0xff00) _chain_intr(old2f);
                else _ax = 0x00ff;
}

// =====================================
// Новый обработчик аппаратного прерывания таймера

void _interrupt _far new8(void) {

        // Вызываем старый обработчик
        (*old8)();
```

```
        // Если была нажата комбинация клавиш Ctrl+PrtSc
        // (iniflag при этом устанавливается в 1
        // новым обработчиком прерывания 9) и
        // если запись в файл уже не началась,
        // то при значении флага критической секции
        // DOS, равном 0, выводим содержимое экрана
        // в файл

        if((iniflag != 0) && (outflag == 0) && *crit == 0) {

                outflag=1;   // Устанавливаем флаг начала вывода
                _enable();   // Разрешаем прерывания

                write_buf(); // Записываем содержимое
                             // буфера экрана в файл

                outflag=0;   // Сбрасываем флаги в исходное
                iniflag=0;   //    состояние
        }
}

// =====================================
// Новый обработчик прерывания 28h, которое вызывает
// DOS, если она ожидает ввода от клавиатуры.
// В этот момент TSR-программа может пользоваться
// функциями DOS.

void _interrupt _far new28(void) {

        // Если была нажата комбинация клавиш Ctrl+PrtSc
        // (iniflag при этом устанавливается в 1
        // новым обработчиком прерывания 9) и
        // если уже не началась запись в файл,
        // то выводим содержимое экрана в файл

        if((iniflag != 0) && (outflag == 0)) {

                outflag=1; // Устанавливаем флаг начала вывода
                _enable(); // Разрешаем прерывания

                write_buf(); // Записываем содержимое видеобуфера
                                    // в файл

                outflag=0;   // Сбрасываем флаги в исходное
                iniflag=0;   //   состояние
        }

        // Передаем управление старому обработчику
        // прерывания 28
        _chain_intr(old28);
}

// =====================================
// Новый обработчик клавиатурного прерывания.
// Он фиксирует нажатие комбинации клавиш Ctrl+PrtSc
// и устанавливает флаг iniflag, который сигнализирует
// о необходимости выбрать подходящий момент и
// записать содержимое видеобуфера в файл

void _interrupt _far new9(void) {

 // Если SCAN-код равен 0x37 (клавиша PrtSc),
 // нажата клавиша Ctrl (бит 4 байта состояния
 // клавиатуры, находящийся в области данных
 // BIOS по адресу 0040:0017 установлен в 1)
 // и если не установлен флаг iniflag,
```

```
 // то устанавливаем флаг iniflag в 1.

 if((inp(0x60) == 0x37) && (iniflag == 0) &&
        (*(char _far *)FP_MAKE(0x40,0x17) & 4) != 0) {

                // Выдаем звуковой сигнал
                BEEP();
                BEEP();
                BEEP();

                _disable(); // Запрещаем прерывания

                // Разблокируем клавиатуру
                // и разрешим прерывания
                _asm {
                        in  al,61h
                        mov ah,al
                        or  al,80h
                        out 61h,al
                        xchg ah,al
                        out 61h,al

                        mov al,20h
                        out 20h,al
                }

                // Устанавливаем флаг запроса
                // на запись содержимого видеобуфера
                // в файл
                iniflag = 1;
                _enable();  // Разрешаем прерывания
 }

 // Если нажали не Ctrl+PrtSc, то
 // передаем управление старому
 // обработчику прерывания 9
 else _chain_intr(old9);

}

// =======================================
// Функция возвращает номер
// текущего видеорежима

int get_vmode(void) {
        char _far *ptr;
        ptr = FP_MAKE(0x40,0x49); // Указатель на байт
                                  // текущего видеорежима
        return(*ptr);
}

// =======================================
// Функция возвращает сегментный адрес
// видеобуфера. Учитывается содержимое
// регистров смещения адреса видеобуфера.

int get_vbuf(int vmode) {
        unsigned vbase;
        unsigned adr_6845;
        unsigned high;
        unsigned low;
        unsigned offs;

        // В зависимости от видеорежима базовый адрес
        // видеобуфера может быть 0xb000 или 0xb800
        vbase = (vmode == 7) ? 0xb000 : 0xb800;
```

```
        // получаем адрес порта видеоконтроллера 6845
        adr_6845 = *(unsigned _far *)(FP_MAKE(0x40,0x63));

        // Считываем содержимое регистров 12 и 13
        // видеоконтроллера
        outp(adr_6845,0xc);
        high = inp(adr_6845+1);

        outp(adr_6845,0xd);
        low = inp(adr_6845+1);

        offs = ((high << 8) + low) >> 4;

        // Добавляем к базовому адресу видеобуфера
        // смещение, взятое из регистров видеоконтроллера
        vbase += offs;

        return(vbase);
}

// =====================================
// Функция возвращает количество символов в строке
// для текущего видеорежима

int get_column(void) {
        return(*(int _far *)(FP_MAKE(0x40,0x4a)));
}

// =====================================
// Функция возвращает количество строк
// для текущего видеорежима

int get_row(void) {
        unsigned char ega_info;
        ega_info = *(unsigned char _far *)(FP_MAKE(0x40,0x87));

        // Если нет EGA, то используется 25 строк,
        // если EGA присутствует, считываем число
        // строк. Это число находится в области данных
        // BIOS по адресу 0040:0084.

        if(ega_info == 0 || ( (ega_info & 8) != 0) ) {
                return(25);
        }
        else {
                return(*(unsigned char _far *)
                        (FP_MAKE(0x40,0x84)) + 1);
        }
}

// =====================================
// Функция записи содержимого видеобуфера в
// файл

int write_buf(void) {

// Видеопамять состоит из байтов символов и байтов
// атрибутов. Нам нужны байты символов chr.
typedef struct _VIDEOBUF_ {
         unsigned char chr;
         unsigned char attr;
} VIDEOBUF;

        VIDEOBUF _far *vbuf;
        int i, j, k, max_col, max_row;
        FILE *out_file;
        char fname[20],ext[8];
```

```
        i=get_vmode(); // Получаем номер текущего
                                // видеорежима

        // Для графического режима ничего не записываем
        if(i > 3 && i != 7) return(-1);

        // Устанавливаем указатель vbuf на видеобуфер
        vbuf=(VIDEOBUF _far *)FP_MAKE(get_vbuf(i),0);

        // Определяем размеры экрана
        max_col = get_column();
        max_row = get_row();

        // Формируем имя файла для записи образа экрана
        itoa(name_counter++,ext,10);
        strcpy(fname,"!grab");
        strcat(fname,ext);
        strcat(fname,".doc");

        out_file=fopen(fname,"wb+");

        // Записываем содержимое видеобуфера в файл
        for(i=0; i<max_row; i++) {
                for(j=0; j<max_col; j++) {

                        fputc(vbuf->chr,out_file);
                        vbuf++;

                }

                // В конце каждой строки добавляем
                // символы перевода строки и
                // возврата каретки
                fputc(0xd,out_file);
                fputc(0xa,out_file);
        }
        fclose(out_file);
        return(0);
}

// =====================================
// Функция выводит на экран инструкцию по
// использованию программы GRAB

int hello(void) {
        printf("\nУтилита копирования содержимого"
               "\nэкрана в файл GRAB<n>.DOC"
               "\nCopyright (C)Frolov A.,1990"
               "\n"
               "\nДля копирования нажмите Ctrl+PrtSc"
               "\n");
}
```

Приведем пример TSR-программы, написанной на языке ассемблера. Эта программа переназначает прерывание 13h, которое используется для работы с дисками. Она позволяет организовать защиту диска от записи. При первом запуске программа включает защиту, при втором выключает, потом опять включает и так далее. В качестве флага - признака включения или выключения защиты, используется компонента смещения вектора прерывания F0h, зарезервированного для интерпретатора BASIC.

```
                .MODEL tiny
                .CODE
                .STARTUP
```

```
                jmp begin

old_int13h_off  dw 0  ; Адрес старого обработчика
old_int13h_seg  dw 0  ; прерывания 13h

old_int2Fh_off  dw 0  ; Адрес старого обработчика
old_int2Fh_seg  dw 0  ; прерывания 2Fh

; Новый обработчик прерывания 2Fh нужен
; для проверки наличия программы в памяти
; при ее запуске для предохранения
; от повторного запуска

new_int2Fh  proc  far
                cmp   ax,0FF00h
                jz    installed

                jmp   dword ptr cs:old_int2Fh_off

; Если код функции 0FF00h, то возвращаем
; в регистре AX значение 00FFh. Это признак
; того, что программа уже загружена в память

installed:
                mov   ax,00FFh
                iret

new_int2Fh  endp

; Новый обработчик прерывания 13h. Для команд записи
; на жесткий диск выполняет проверку содержимого
; компоненты смещения вектора прерывания FFh.
; Эта ячейка служит для триггерного переключения
; режима работы прерывания 13h - включения/выключения
; защиты записи.

new_int13h  proc  far
                cmp   ah,3 ; запись сектора
                je protect
                cmp   ah,5 ; форматирование трека
                je protect
                jmp   dword ptr cs:old_int13h_off

old_int13h:
                pop   es
                pop   bx
                pop   ax

                jmp   dword ptr cs:old_int13h_off

protect:
                push  ax
                push  bx
                push  es

; Проверяем значение триггерного флага защиты
                xor   ax,ax
                mov   es,ax
                mov   bx,0F0h*4
                mov   ax,WORD PTR es:[bx]
                cmp   ax,0FFFFh

                jne   old_int13h

; Для флоппи-дисков защиту не включаем

                cmp   dl,0
```

```
                je old_int13h
                cmp     dl,1
                je old_int13h

                pop   es
                pop   bx
                pop   ax

; Имитируем ошибку записи при попытке
; записать данные на защищенный от записи диск

                mov   ah,3
                stc
                ret 2

new_int13h  endp

;==============================
; Точка входа в программу

begin proc  far

; Проверяем, не загружена ли уже программа
; в память

                mov   ax,0FF00h
                int   2Fh

                cmp   ax,00FFh
                jne   first_start

                jmp   invert_protect_flag

; Первоначальный запуск программы

first_start:

; Устанавливаем триггерный флаг защиты записи
; в состояние, соответствующее включенной защите

                xor   ax,ax
                mov   es,ax
                mov   bx,0F0h*4
                mov   WORD PTR es:[bx],0FFFFh

; Запоминаем адрес старого обработчика прерывания 13h

                mov   ax,3513h
                int   21h
                mov   cs:old_int13h_off,bx
                mov   cs:old_int13h_seg,es

; Запоминаем адрес старого обработчика прерывания 2Fh

                mov   ax,352Fh
                int   21h
                mov   cs:old_int2Fh_off,bx
                mov   cs:old_int2Fh_seg,es

                push  cs
                pop      ds

; Выводим сообщение о включении защиты

                mov   dx,offset msg_on
                mov      ah,9
                int   21h
```

```
; Устанавливаем новые обработчики прерываний 13h и 2Fh

                mov   dx,OFFSET new_int13h
                mov      ax,2513h
                int   21h

                mov   dx,OFFSET new_int2Fh
                mov   ax,252Fh
                int   21h

; Завершаем программу и оставляем резидентно
; в памяти часть программы, содержащую новые
; обработчики прерываний

                mov   dx,OFFSET begin
                int   27h

; Если это не первый запуск программы,
; инвертируем содержимое триггерного флага защиты

invert_protect_flag:

                xor   ax,ax
                mov   es,ax
                mov   bx,0F0h*4

                mov   ax,WORD PTR es:[bx]
                not   ax
                mov   WORD PTR es:[bx],ax
                mov   cx,ax

                cmp   cx,0FFFFh
                je prot_on

; Выводим сообщение о выключении защиты

                mov   dx,OFFSET msg_off
                jmp   short send_msg
prot_on:
; Выводим сообщение о включении защиты
                mov   dx,OFFSET msg_on
send_msg:
                mov     ah,9
                push    cs
                pop     ds
                int   21h

                .EXIT
begin endp

msg_on      db 'Защита диска включена$'
msg_off     db 'Защита диска ВЫКЛЮЧЕНА$'
                end
```

Этим примером мы завершим обзор TSR-программ. В следующей главе будет описан другой вид резидентных программ - драйверы. Использование драйвера - более предпочтительный, чем TSR-программы способ организовать обслуживание нестандартной аппаратуры.