Фирма «РУСИЧ»

TR-DOS

АДАПТАЦИЯ ПРОГРАММ НА ДИСК В СИСТЕМЕ TR-DOS

НТК Системотехник 1991

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Кто-то спросит: «А зачем мучиться, переводя программы из ленточного формата в дисковый, если интерфейс Beta-Disk оснащен замечательной кнопкой Magic Button (волшебная кнопка, англ.), позволяющей адаптировать программы одним пальцем? И не беда, что после такой адаптации программу приходится запускать с помощью GO TO». Не вступая в долгие споры, просто опишу ужасы, которые сотворяет с программой эта воистину Волшебная Кнопка.

Под внешним благополучием программ, «адаптированных» Волшебной Кнопкой*, может скрываться огромное количество гадостей. Согласитесь, неприятно, когда программа начинает сбоить в самом интересном месте, и Вы после нескольких часов игры, наконец, почти добравшись до заветной цели, вдруг получаете дулю!

А происходит вот что. Как только Ваша рука прикасается к кнопке:

- в машинный стек компьютера записывается большой массив данных (сохраняются значения всех регистров процессора и т. п.), а этого уже достаточно, чтобы испортить часть программы;
- подпрограмма, обслуживающая кнопку, бесцеремонно портит несколько ячеек системных переменных бейсик-интерпретатора; по всей видимости, авторы операционной системы TR-DOS забыли, что коммерческие игровые программы часто используют эти ячейки для совсем других целей;
- происходит самое очевидное безобразие — портятся заставки программ.

Программ, которые кнопка не портит, совсем немного, отыскать их можно, в основном, среди самых древних. Среди программ последних лет, которые пишутся с помощью мощных ЭВМ, программ, где на счету каждый бит, где все выверено настолько, что комар носа не подточит, — найти такую, которую не испортишь кнопкой, довольно непросто.

Подумайте и решите для себя, что лучше: иметь после некоторых усилий доброкачественный продукт или уродство, запускаемое оператором GO TO?

---

* В дальнейшем, с Вашего позволения, буду называть ее просто кнопкой.

Глава 1

КАК БОРОТЬСЯ С БЕЙСИКОМ

Загрузчики программ, с которыми Вы можете встретиться, подразделяются на три категории: *загрузчики на Бейсике* (загрузка производится оператором LOAD), *загрузчики в машинных кодах* и *смешанные загрузчики*.

Проще и приятнее, безусловно, иметь дело с загрузчиками на Бейсике. Их можно разделить на *открытые* и *закрытые*.

Открытые загрузчики — это когда после загрузки первого файла программы (то есть собственно загрузчика) нажимаешь **Break**, затем **Enter**, и на приятном голубом фоне появляется:

```
10 INK 1: PAPER 5: BORDER 5: CLEAR 25199
20 LOAD ""SCREEN$: LOAD ""CODE
30 RANDOMIZE USR 32768
```

или что-то в этом роде.

Надеюсь, что не ущемлю Ваше самолюбие, если все же покажу, как адаптировать к диску такую программу:

```
10 INK 1: PAPER 5: BORDER 5: CLEAR 25199
20 RANDOMIZE USR 15619: REM :LOAD "name1"CODE 16384
25 RANDOMIZE USR 15619: REM :LOAD "name2"CODE
30 RANDOMIZE USR 32768
```

– где name1 и name2 — имена, которые Вы присвоите файлам, следующим за загрузчиком. Иногда эти имена можно оставить прежними, но чаще их приходится изменять. Во-первых, имена файлов в кассетной версии могут содержать символы или комбинации символов, которые будут восприниматься системой TR-DOS неправильно. Например, встретив оператор LOAD "B:SCREEN", TR-DOS будет загружать файл "SCREEN" с дисковода "B", а отнюдь не файл с именем "B:SCREEN", как хотелось бы. Во-вторых, в отличие от кассеты, на которой файлов с именем "SCREEN" может находиться сколько угодно, на диске файл с таким именем должен быть только один. Поэтому обычно файлы программы называют именами, производными от имени самой программы (например, "rambo$", "rambo cd" и т. п.). Кроме того, считается хорошим тоном в имени основного файла (загрузчика) использовать прописные буквы, а в именах остальных — строчные*.

---

* На самом деле, последнее придумал я сам, но, по-моему, это можно считать хорошим тоном (*прим. авт.*).

Изменить имена файлов на желаемые можно еще до перенесения программы на диск, воспользовавшись обыкновенным ленточным копировщиком (TF COPY, COPY DE LUXE, COPY COPY и др.), после чего останется лишь перебросить файлы на диск специальным копировщиком «лента–диск». Удобнее всего пользоваться копировщиками Programmable Copier v. 2.0 (PCopier), Pcopier Plus или AMCopier. При работе с перечисленными копировщиками, во-первых, можно поменять имена файлов уже после копирования файлов на диск, а во-вторых, есть уверенность, что файлы и диск не будут испорчены (при использовании же копировщика Дерещука такой уверенности нет).

Но вот Вы, самым тщательным образом просмотрев все свои диски, убедились, что «специального копировщика» на них нет. Не отчаивайтесь, еще не все потеряно. Загрузите программу в обыкновенный копировщик и внимательно изучите все, что он расскажет о ее файлах. Запишите длины и адреса загрузки каждого файла и смело нажимайте кнопку сброса. Теперь, обладая столь ценной информацией, можно действовать дальше.

Пусть, например, копировщик сообщил следующее:

файл "SCREEN" — адрес загрузки 49152, длина 6912;
файл "CODE" — адрес загрузки 25200, длина 40335.

Используя эти данные, можно на основе приведенного варианта загрузчика написать специальный копировщик:

```
10 INK 1: PAPER 5: BORDER 5: CLEAR 25199
20 LOAD ""SCREEN$
21 RANDOMIZE USR 15619:
   REM : SAVE "shaky sc"CODE 16384,6912
30 LOAD ""CODE
31 RANDOMIZE USR 15619:
   REM : SAVE "shaky mn"CODE 25200,40335
40 STOP
```

– то есть после каждого оператора загрузки файла с ленты нужно поставить оператор записи на диск, указав адрес и длину файла. Если в ленточном загрузчике не упоминается адрес загрузки, то используйте адрес, который любезно сообщил копировщик. Но будьте внимательны, не всегда адрес в загрузчике указан в явном виде: в приведенном примере копировщик указал адрес первого файла 49152, но загрузка производится оператором LOAD ""SCREEN$, который является эквивалентом LOAD ""CODE 16384,6912. Поэтому файл необходимо записывать с адреса 16384, а не 49152.

На этом, как правило, Ваши мучения должны закончиться. Единственной «серьезной» проблемой при переделке подобного загрузчика может стать то, что при выводе его листинга цвет фона и тона совпадут, и Вам придется изменить один из цветов оператором INK или PAPER. Но даже этого можно избежать, подгрузив файл загрузчика оператором MERGE, а не LOAD. В этом случае программа не запустится, а появится сообщение 0 OK, 0:1. Единственно, не следует

забывать, что делать это можно только на чистой машине, то есть после нажатия клавиши сброса.

Надеюсь, что в вопросах, касающихся открытых загрузчиков, наступила полная ясность даже у того, кто ничего не понял.

Ваше счастье, если просмотрев листинг загрузчика, Вы увидите нечто похожее на предыдущий пример, то есть на нормальную бейсик-программу, но на практике гораздо чаще появляется что-нибудь вроде:

```
0 REM Look here, little bozo, why don't you go fuck yourself?
0 DRAW USR VAL "24500",deBillGilbert'89 0 REM FLASH
  DRAW ???Aw COPY M
```

или еще более страшное. Подобный ужас можно назвать з а к р ы т ы м загрузчиком. Попытаюсь научить Вас «открывать» его.

Бейсик-программу защищают от просмотра и редактирования несколькими способами. Рассмотрим самые популярные из них.

Вы, вероятно, уже сталкивались с таким фокусом, как нулевая строка, которую нельзя ни удалить, ни вызвать на редактирование. Справиться с такой защитой можно двумя путями. Первый заключается в перенумерации строк с помощью одной из многочисленных сервисных программ (ZXED, например). После чего обычно получается вполне доброкачественный листинг (отсутствуют нулевые строки и т. п.).

Второй путь более изящен, но эффективен только в том случае, когда нулевая строка одна и располагается первой в программе. Для его реализации нужно иметь представление о формате хранения строк бейсик-программы в памяти ZX Spectrum*:

номер строки (2 байта)	длина строки (2 байта)	операторы Бейсика	перевод строки

Вы уже, вероятно, поняли, что изменить номер строки проще простого, записав оператором POKE требуемое число в первые два байта программы. Но куда именно записывать? Вот этим вопросом мы сейчас и займемся.

Определить начало бейсик-программы поможет системная переменная PROG**. Она находится по адресу 23635 и занимает два байта. Таким образом, выяснить адрес первого байта первой строки бейсик-программы можно, выполнив оператор

```
PRINT PEEK 23635+256*PEEK 23636
```

Если к Вашему Speccy не подключено никакой периферии, то после выполнения этого оператора на экране появится число 23755. Если

---

* Более подробно об этом см. [1].

** Подробнее о системных переменных читайте в [1].

подключен и инициализирован Beta-Disk, то появится число 23867, поскольку TR-DOS резервирует 112 ячеек для своих нужд.

Итак, адрес, по которому хранится номер начальной строки бейсик-программы, обнаружен: пусть, к примеру, он равен 23755. Теперь можно менять номер первой строки:

```
POKE 23755,N-256*INT (N/256): POKE 23756,INT (N/256)
```

– где N — требуемый номер. Это идеальный способ переопределения номера строки. Можете, кстати, поэкспериментировать, задавая номера строк более 9999: получаются весьма любопытные результаты. Как показывает практика, необходимость изменять номера более 255 возникает редко. Поэтому обычно достаточно выполнить всего один оператор:

```
POKE 23756,N
```

– где N — номер строки не более 255. Этим оператором изменяется только младший байт номера строки (еще раз хочу напомнить, что последние два примера приведены, исходя из того, что бейсик-программа начинается с адреса 23755).

Выше было сказано, что такой способ борьбы с нулевой строкой эффективен только, когда нулевая строка находится в начале программы. На самом же деле, ничто не мешает изменить номер второй и последующих строк. Для этого пригодится информация о длине строки, содержащаяся в двух следующих за номером строки байтах. Прибавив длину строки к адресу ее начала, получим адрес следующей строки.

Не менее распространенным приемом защиты бейсик-программ является добавление в строки бейсик-программы различных управляющих кодов: INK, PAPER, AT, TAB и других. Полную таблицу контрольных кодов Вы сможете найти в [1, 2].

Для примера поясню разницу между контрольным кодом INK и оператором INK. Если при выводе листинга программы бейсик-система встречает оператор INK, она честно представляет его на экране, не производя никаких действий. Если же встречается контрольный код INK, то следующий за ним байт бейсик-система воспринимает как код цвета тона и продолжает вывод листинга этим цветом. Контрольный код INK и оператор INK различаются своими кодами в таблице символов (см. [1, 2]), и их нельзя путать. Все сказанное справедливо для PAPER, FLASH, TAB и др. Необходимо учесть только, что позиция вывода на экран задается двумя байтами, следующими за контрольным кодом AT и TAB, а не одним, как для других контрольных кодов.

Большинство контрольных кодов можно удалить клавишей **Delete**, вызвав бейсик-строку на редактирование. Поочередно нажимайте клавиши «курсор вправо» и **Delete**, и если авторы защиты использовали к о р р е к т н ы е значения контрольных кодов, то Вы без осложнений рано или поздно получите на экране вполне осмысленную информацию. Но бывает и так: когда, например, бейсик-интерпрета-

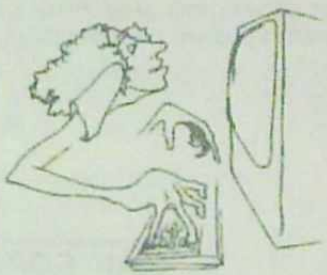
тор просят вывести листинг программы 121-м цветом в знакоместо с координатами (240,86), тогда он просто-напросто прекращает вывод листинга на экран, а при попытке редактирования строки начинает истошно орать. В этом случае нужно сначала «заткнуть ему рот» оператором POKE 23608,0 и далее действовать по приведенной схеме.

Еще одна крайне неприятная вещь, которая может помешать справиться с программой при адаптации ее на диск, — это нехватка памяти. Она может возникнуть из-за того, что, во-первых, TR-DOS резервирует 112 байт для своих системных переменных; во-вторых, операторы загрузки с диска могут занимать в памяти значительно больше места, чем аналогичные инструкции для ленты (если Вы привыкли исчислять память мегабайтами, слово «значительно» беру назад); в-третьих, системе TR-DOS необходим буфер не менее 256 байт для записи/чтения файлов, вывода каталога и т. п. и еще больше для распечатки каталога дискеты командой LIST. Этого дополнительного расхода памяти бывает достаточно, чтобы компьютер выдал сообщение: Out of memory, No room for line или еще что-нибудь в этом роде.

Не печальтесь, в этом случае (как, впрочем, и во всех остальных) компьютер можно обмануть. Наиболее простой и доступный способ — это разделение загрузчика на две или более части.

Допустим, исходный ленточный загрузчик выглядел так:

```
10 CLEAR 24250
20 LOAD ""SCREEN$
30 LOAD ""CODE
40 LOAD ""CODE 23296,256
50 RANDOMIZE USR 23296
60 LOAD ""CODE
70 RANDOMIZE USR 24576
```



Без толики сомнения Вы написали дисковый загрузчик для этой программы:

```
10 CLEAR 24250
20 RANDOMIZE USR 15619:
   REM : LOAD "bomb sc"CODE 16384
30 RANDOMIZE USR 15619: REM :LOAD "bomb mn"CODE
40 RANDOMIZE USR 15619:
   REM : LOAD "bomb pb"CODE 23296
50 RANDOMIZE USR 23296
60 RANDOMIZE USR 15619: REM : LOAD "bomb ls"CODE
70 RANDOMIZE USR 24576
```

и, запустив программу, вместо игры получили сообщение 4 Out of memory, 20:1. Только не вздумайте тянуться к @безьяне (так некоторые именуют кнопку Magic, поскольку в простонародье символ @, которым TR-DOS помечает названия файлов, «сброшенных» кнопкой Magic, часто называют «обезьяной» или «собакой». Однако, в целом,

звучит неплохо и соответствует истинному положению вещей). Попробуйте разбить загрузчик на две части. Например, в нашем случае первая часть загрузчика (назовем этот файл "BOMB") будет выглядеть так:

```
10 CLEAR 24250
20 RANDOMIZE USR 15619:
   REM : LOAD "bomb sc"CODE 16384
30 RANDOMIZE USR 15619: REM : LOAD "bomb mn"CODE
40 RANDOMIZE USR 15619: REM : LOAD "bomb bs"
```

Вторую часть назовем "bomb bs":

```
10 RANDOMIZE USR 23296
20 RANDOMIZE USR 15619: REM : LOAD "bomb ls"CODE
30 RANDOMIZE USR 24576
```

Компьютер загрузит сначала первый загрузчик "BOMB", которому места в памяти вполне достаточно. Затем, после выполнения заданных операций, первый загрузчик загрузит второй ("bomb bs"), ему, в свою очередь, места также хватает. Второй загрузчик закончит эпопею с загрузкой и запустит программу.

Могут возникать ситуации, когда приходится «распиливать» загрузчик на три или более части, однако это бывает крайне редко. Точнее, крайне редко такой прием помогает.

Глава 2

КАК БОРОТЬСЯ С BREAK'ом

В предыдущей главе было подробно описано, как обращаться с загрузчиком на Бейсике уже после того, как был получен его листинг. Даже в том случае, когда полученное «нечто» слабо напоминает листинг. Вполне законно у Вас мог возникнуть вопрос, как получить это «нечто» (листинг), ведь далеко не всегда, нажав клавишу **Break** после загрузки первого файла, Вы получите то, что хотели. Результаты вмешательства в нормальный ход загрузки могут быть совершенно непредсказуемыми, начиная с того, что программа сбросится, и кончая тем, что она повиснет или обзовет Вас неприличным словом.

Подробнее опишу эффекты, возможные при останове программы клавишей **Break**, а также способы, которыми эти эффекты достигаются. Эта информация может быть полезна скорее не для вскрытия чужих программ, а для защиты своих (не забывайте, однако, что воспользовавшись данными ниже указаниями, Вы защитите программу только от того, кто не читал эту брошюру).

1. После нажатия на клавишу **Break** загрузчик продолжает исправно работать.

Перекреститесь и откажитесь от мысли вскрыть эту программу — к этому Вы еще не готовы, так как гарантированно можно заявить, что загрузчик написан в машинных кодах (это не имеет отношения к профессионалам, которые читают это произведение только для того, чтобы посмеяться над автором).

2. После нажатия **Break** компьютер обнуляется.

Если загрузчик написан на Бейсике, то наиболее вероятно, что в нем использована следующая инструкция:

```
POKE (PEEK 23613+256*PEEK 23614),0:
POKE (PEEK 23613+256*PEEK 23614)+1,0
```

или аналогичная.

Выполнение подобной операции изменяет адрес «возврата по ошибке» на нулевой (системная переменная ERR_SP). Таким образом, вместо того, чтобы перейти на подпрограмму ПЗУ, которая выводит сообщение об ошибке, программа переходит на нулевой адрес: машина обнуляется (те, кто ни разу не видел, как машина обнуляется, могут попробовать ввести RANDOMIZE USR 0 или нажать кнопку сброса).

3. После нажатия **Break** экран становится черным и машина ни на что не реагирует (кроме кнопки сброса, надо полагать). Скорее всего использован оператор POKE 23659,0, устанавливающий количество строк в служебном экране равным нулю. К описанному эффекту приводит попытка бейсик-интерпретатора уложить свое сообщение в ноль строк.

4. После нажатия **Break** цвет экрана не меняется, нижние две строки экрана имеют цвет бордюра. Цвет экрана также может иметь цвет бордюра (в том числе и черный — не путайте с предыдущим случаем). При нажатии на клавиши слышится нежное пощелкивание, но на экране ничего не изменяется.

Если Вы еще не успели пощелкать клавишами, наберите BORDER 7 (или любое другое значение) и, невзирая на то, что на экране ничего не появляется, нажмите **Enter**. После этого Вы скорее всего увидите надпись 0 OK, 0:1. Теперь можно работать как обычно. (Если Вы уже успели пощелкать, то предварительно пощелкайте клавишей **Delete**).

В этом случае использовался оператор POKE 23624,n, где n — число, задающее атрибуты служебного экрана, а 23624 — адрес системной переменной BORDCR.

Нажатие клавиши **Break** — довольно невкусный способ раскрутки программ. Значительно приятней лишить загрузчик автостарта: тогда Вы получаете полную свободу действий.

Самое простое, что можно сделать — это загрузить копировщик COPY86/M или Out Copy (что, в общем-то, одно и то же) и скопировать загрузчик, не забыв перед записью нажать клавишу R. Таким образом, Вы получите программу без автостарта.

Подобные вещи позволяют проделывать и другие копировщики, например, написанные Tadeusz'ем Wilczek'ом (COPY NEW, NEW FORMAT COPY, COPY COPY, COPY COPY COPY и т. п.).

Но допустим, у Вас не нашлось копировщика, позволяющего снимать программы автостарта. Попробую помочь. Наберите следующую программу:

```
10 FOR A=23296 TO 23334
20 READ S: POKE A,S
30 NEXT A
40 RANDOMIZE USR 23296
50 STOP
60 DATA 221,33,0,128,17,17,0,62,0,55,205,86,5,
   33,0,200,34,11,128,62,127,219,254,203,71,32,-8,
   221,33,0,128,17,17,0,62,0,195,194,4
```

Когда Вы ее запустите, машина войдет в режим загрузки, но загрузит только заголовок. Затем, при нажатии на клавишу **Space**, программа запишет на ленту новый заголовок — без автостарта.

Для тех, кто слышал слово «ассемблер» и на вопрос: «Который час?» вместо «Восемь тридцать» отвечает: «Два эф девятого», привожу исходный текст с комментариями:

```
        ORG   23296      ; 5B00h
        LD    IX,32768   ; 8000h адрес загрузки заголовка
        LD    DE,17      ; 11h длина заголовка
        LD    A,0        ; флажок "заголовок"
        SCF              ; флажок "загрузка"
        CALL  1366       ; 0556h загрузка файла
        LD    HL,33024   ; 8100h задание строки автостарта
        LD    (32779),HL
WAIT    LD    A,127      ; 7Fh ожидание нажатия Space
        IN    A,(254)
        RRA
        JR    C,WAIT
        LD    IX,32768   ; см. выше
        LD    DE,17
        LD    A,0
        CALL  1218       ; 04C2h запись на ленту
        RET
```

Получив живой загрузчик, попытайтесь его идентифицировать в рамках того, что Вам уже известно.

Как говорилось в начале первой главы, загрузчики бывают на Бейсике, в машинных кодах и смешанные. Предварительную иденти-

фикацию загрузчика можно произвести, изучив файлы программы. Бейсик позволяет загружать только файлы с заголовком. Загрузчик в машинных кодах может загружать что угодно и как угодно, но, как правило, загружает файлы без заголовков (в оригинальных версиях игровых программ загружается такое, перед чем пасует любой нормальный копировщик: файлы этих программ не копируются). Таким образом, если загружаются файлы без заголовка, то определенно можно сказать, что загрузчик, с которым Вы имеете дело, написан в машинных кодах. Если все файлы имеют заголовок, то можно предполагать, что загрузчик написан на Бейсике, но определенно сказать этого пока нельзя. Если файлы попадаются как с заголовком, так и без него, то можно предполагать что угодно, кроме того, что загрузчик на Бейсике.

Нередки случаи, когда загрузчик, написанный на Бейсике, загружает и запускает загрузчик в кодах.

Для окончательной идентификации необходимо изучить содержимое загрузчика. Если все файлы программы имеют заголовок, и количество операторов LOAD совпадает с количеством файлов, то можете быть спокойны, загрузчик почти наверняка на Бейсике.

Глава 3

ЕЩЕ О БЕЙСИКЕ

Программист, не знакомый с представлением чисел бейсик-интерпретатором компьютера ZX Spectrum, немало удивится, столкнувшись со следующей программой:

```
10 CLEAR 100
20 PAPER 27: INK 8E-12: BORDER 65536: CLS
30 PRINT AT -10000,22E22;"PLEASE WAIT"
40 LOAD ""CODE 0,0
50 RANDOMIZE USR 0
```

Такой программист будет настоятельно утверждать, что этот бред написан не иначе, как пациентом учреждения на Пряжке, до тех пор, пока не запустит программу и не увидит результаты ее работы. Что произойдет с программистом после того, как в середине экрана появится надпись PLEASE WAIT, затем загрузится и запустится игра, предсказать трудно. Вероятнее всего, он побежит в ПНД с предполагаемым диагнозом «маниакально-депрессивный психоз на почве программирования».

Для того чтобы с Вами этого не произошло, раскрою Вам одну тайну. Все дело в том, что при обработке чисел бейсик-интерпретатор использует два их представления. Первое представление — это

приятная для глаз символьная форма, в которой число 16389, например, будет записано пятью байтами согласно таблице ASCII. Однако такая форма относительно долго обрабатывается бейсик-интерпретатором, посему для своих нужд он использует другое представление числа — двоичное. Двоичная форма числа, предваряемая префиксом — кодом 14 (0Eh), хранится в памяти непосредственно после символьной формы. Число 16389, например, выглядит в памяти следующим образом:

	символьная форма					пре-	двоичная форма					
	1	6	3	8	9	фикс						
...	49	54	51	56	57	14	0	0	5	64	0	...

Отсюда видно, что двоичная форма числа занимает 5 байт плюс байт префикса. При записи целых констант в промежутке от 0 до 65535 «значащими» являются 3-й и 4-й байты после префикса. В этом случае двоичную форму числа можно перевести в десятичную с помощью оператора

```
PRINT PEEK (n+3)+256*PEEK (n+4)
```

— где n — адрес, по которому хранится байт префикса.

Двоичная и символьная формы числа существуют параллельно. При выполнении инструкций бейсик-интерпретатор руководствуется только двоичной формой, символьная форма его не интересует, она сохраняется ради удобочитаемости программ. Поэтому «порча» символьного представления чисел приводит в ужасный вид листинг программы, но не влияет на ее работоспособность.

Двоичная форма числа добавляется в момент проверки синтаксиса строки, то есть по окончании набора строки и нажатии клавиши **Enter**. При вызове на редактирование строки двоичная форма удаляется, а после ввода исправленной строки восстанавливается в соответствии с новой редакцией.

Из сказанного должно быть понятно, что загадочная программа, приведенная в начале этой главы, не будет работать, если ее ввести с клавиатуры обычным способом, не используя каких-либо специальных приемов (быть может, Вы уже попробовали ее набрать и запустить и решили, что не кто иной, как я, и являюсь пациентом вышеуказанной клиники).

Для того, чтобы привести программу к такому извращенному виду, при этом оставив ее работоспособной, необходимы внешние средства, например, монитор-отладчик MONS4 или подобные. Однако мы пока ведем разговор об адаптации программ на диск, а не о приведении их в состояние алкогольного опьянения, поэтому задачи у нас стоят обратные.

Можно было бы заняться пространным описанием представления чисел в двоичной форме, но я предпочту дать практические советы по вычислению того, что скрыто под маской символьной формы.

Наиболее простым и эффективным способом определения истинных значений чисел, по моему опыту, является подстановка оператора **PRINT** вместо операторов программы. В этом случае все числа, используемые программой, будут последовательно выведены на экран в нормальном виде.

Если приведенную в начале главы программу изменить следующим образом (обратите внимание, что не использующие чисел операторы и другие «лишние» символы заменены пробелами, обозначенными в данном случае знаком подчеркивания):

```
10 PRINT 100
20 PRINT 27: PRINT 8E-12: PRINT 65536:__
30 PRINT _-10000,22E22;_______________
40 PRINT ___0,0
50 PRINT _0
```

– то в результате ее выполнения на экране появится:

```
24999
0
4
0
10        10
25000   40000
25000

0 OK, 50:1
```

Подставив полученные значения в программу, получим привычный и понятный текст:

```
10 CLEAR 24999
20 PAPER 0: INK 4: BORDER 0: CLS
30 PRINT AT 10,10;"PLEASE WAIT"
40 LOAD ""CODE 25000,40000
50 RANDOMIZE USR 25000
```

Если Вы уже потянулись к клавише **Edit**, чтобы побыстрее расставить операторы **PRINT**, советую Вам прочесть все с начала. Как уже говорилось, при вызове на редактирование двоичная форма числа безвозвратно теряется, остается только символьная, и если Вы испра-

вите INK 8E−12 на PRINT 8E−12 с помощью бейсик-редактора, то и получите не что иное, как 8E−12. Чтобы заменить операторы программы на PRINT, не «сломав» двоичной формы чисел, нужны другие способы. Можно воспользоваться все тем же отладчиком MONS4; в крайнем случае сойдет и оператор POKE. В этой брошюре я ставлю своей целью обучать работе с отладчиком, поэтому расскажу лучше, как победить программу без подручных средств.

Оператор POKE очень хорош, но только тогда, когда знаешь, какие параметры должны следовать после него. А для этого нужно заглянуть в память компьютера. Можно, например, воспользоваться строчкой, которая выводит на экран адрес и содержимое ячеек памяти и символьное представление этого содержимого:

```
FOR n=23755 TO 4E10: PRINT n,PEEK n;TAB 22;
   CHR$ (PEEK n+(PEEK n=32)): NEXT n
```

Выполнив эту строку, на экране получим:

```
23755  0     ?
23756  10    ?
23757  11    ?
23758  0     ?
23759  253   CLEAR
23760  49    1
23761  48    0
23762  48    0
23763  14    ?
23764  0     ?
23765  0     ?
23766  167   COS
23767  97
23768  0     ?
23769  13    ?
23770  0     ?
23771  20    ?
23772  38    &
23773  0     ?
23774  218   PAPER
23775  50    2
23776  55    7

scroll?
```

номер строки
длина строки
ключевое слово
символьная форма параметра
префикс
двоичная форма параметра
возврат каретки
номер строки

Теперь нужно внимательно изучить содержимое экрана.

Как Вы уже знаете, первые два байта — это номер строки, следующие два — длина строки, далее ключевое слово CLEAR, которое, собственно, нам и нужно. Берем листок бумаги и карандаш и записываем адрес, по которому находится CLEAR (23759), и продолжаем сей

высокоинтеллектуальный труд. После CLEAR следует символьная форма числа, затем двоичная, далее — символ «возврат каретки» (конец строки), номер и длина следующей строки и оператор PAPER. Опять берем карандаш и записываем адрес оператора PAPER.

Просмотрев программу до конца и записав все адреса, которые необходимо заменить, можно приступать к изменениям:

```
POKE 23759,245: REM Заменяем CLEAR
POKE 23774,245: REM Заменяем PAPER
POKE 23884,245: REM Заменяем INK
POKE 23997,245: REM Заменяем BORDER
POKE 24110,32:  REM Заменяем CLS
POKE 24117,32:  REM Заменяем AT
```

и т. д. Число 245 — это код ключевого слова PRINT, а 32 — код пробела. В данном примере пробелами заменяются следующие операторы и символы:

```
CLS
AT
"PLEASE WAIT"
""CODE
USR
```

Когда будут сделаны все необходимые изменения, просмотрите листинг программы, дабы убедиться в том, что исправлено все, что нужно. Затем программу можно запустить и изучать полученные результаты.

Иногда эту операцию можно несколько упростить. Если в двоичном представлении чисел используются целые константы от 0 до 65535, то определить их истинные значения можно непосредственно при просмотре памяти. Для этого желательно добавить к карандашу и бумаге калькулятор.

Найдите в памяти символьную форму числа, которое нужно «разоблачить»; за ней следует префикс и двоичная форма. Если число целое в промежутке от 0 до 65535, то после префикса будут следовать два нуля, затем два числа в интервале от 0 до 255 и еще один ноль. Если это не так, то определенно можно сказать, что число либо не целое, либо выходит за пределы указанного диапазона. В этом случае проще прибегнуть к предыдущему способу. Если число «подходит», то умножьте 4-й после префикса байт на 256 и прибавьте 3-й (см. стр. 14). В рассматриваемом примере, чтобы получить параметр оператора CLEAR, нужно вычислить:

```
PRINT PEEK (23763+3)+256*PEEK (23763+4)
```

или

```
PRINT 167+256*97
```

Полученный результат (24999) и есть то, что от нас пытались утаить.

Раскроем секрет еще одного распространенного фокуса, основанного на знании формата записи чисел бейсик-интерпретатором (хотя, возможно, отгадку Вы уже знаете).

Вам, наверняка, встречались подобные строки:

```
10 CLEAR VAL "24999": INK VAL "7": PAPER BIN:
   BORDER BIN
```

Таким образом программисты экономят оперативную память. Как ни парадоксально, но запись VAL "24999" занимает меньше места, чем просто 24999. Так как в первом случае число 24999 заключено в кавычки, то оно является символьной строкой, а не числом и не имеет после себя шести байт двоичного представления. При выполнении функции VAL эта строка переводится в число. Таким образом экономится три байта. Что же касается BIN, то выигрыш еще более очевиден. Если вместо BIN подставить 0, то памяти будет занято на шесть байт больше, а результат — тот же.

Этот способ экономии памяти несколько замедляет выполнение программы, но его можно и нужно использовать в загрузчиках, так как именно они, как правило, наиболее критичны к размеру и не критичны к скорости выполнения.

Глава 4

ЗАГРУЗЧИКИ В МАШИННЫХ КОДАХ

для тех, кто не знает, что такое машинный код

Несмотря на то, что знание ассемблера не является обязательным для понимания того, о чем я буду писать дальше, оно отнюдь не повредит. Еще более полезным является умение пользоваться каким-либо монитором-дизассемблером. Если Вы не знаете, что такое дизассемблер, рекомендую воспользоваться программой из пакета Ultraviolet/Infrared. Это первые ассемблер и дизассемблер, появившиеся на рынке для ZX Spectrum. Программа проста в обращении и вполне приемлема для решения тех задач, которые будут разбираться в этой главе. Позднее Вам придется перейти к чему-нибудь более серьезному (например, MONS4).

Изложить последовательно и понятно организацию загрузки файлов из машинного кода для неподготовленного пользователя — задача более чем сложная, но попробовать можно.

Начнем с самого простого. Предположим, что на ленте имеется программа, состоящая из четырех файлов: файл на Бейсике, кодовый файл и два файла без заголовка. Изучив бейсик-программу, Вы

пришли к выводу, что она загружает кодовый файл и запускает его с адреса 65000.

С помощью копировщика Вы определили, что кодовый файл загружается с адреса 65000 и имеет длину 30 байт, длина файлов без заголовка равна соответственно 6912 и 32768 байт.

Теперь загрузите кодовый файл туда, где ему положено находиться, но запустите не его, а Ваш дизассемблер. Дизассемблировать память необходимо, начиная с адреса запуска, то есть с адреса 65000. Предположим, что дизассемблер показал следующее*:

```
65000  LD    IX,16384    ; стартовый адрес
65004  LD    DE,6912     ; длина
65007  LD    A,255
65009  SCF
65010  CALL  1366        ; вызов подпрограммы загрузки
65013  LD    IX,25000    ; стартовый адрес
65017  LD    DE,32768    ; длина
65020  LD    A,255       ; вызов подпрограммы загрузки
65022  SCF
65023  CALL  1366
65026  JP    40000       ; запуск программы
65029  NOP
```

Не берусь объяснять смысл всех этих закорючек. Если Вы знаете, что они обозначают — Ваше счастье, если нет, то и Бог с Вами. Чтобы мало-мальски разбираться с загрузчиками, необходимо запомнить, что после LD IX следует адрес загрузки файла, после LD DE — его длина. Мнемоники LD A,255 и SCF устанавливают системные регистры, а CALL 1366 вызывает подпрограмму загрузки с ленты. Адрес после CALL может меняться в зависимости от того, использует ли загрузчик стандартную подпрограмму загрузки, расположенную в ПЗУ (как в данном примере), или свою собственную.

В приведенном выше примере мы видим две группы команд, обеспечивающих загрузку файла, и команду JP 40000. Последняя является аналогом оператора RANDOMIZE USR 40000 в загрузчике на Бейсике, то есть запускает основную программу.

Таким образом, мы узнали, что наш загрузчик загружает два файла, первый из которых (длиной 6912 и адресом загрузки 16384), очевидно, — заставка, а второй (32768 и 25000 соответственно) — основной файл программы (обратите внимание, что данные о длинах файла совпадают с данными, предоставленными копировщиком); кроме того, мы узнали, что программа запускается с адреса 40000.

Получив эти данные, можно приступить к адаптации программы.

Пожалуй, самое сложное, что предстоит сделать, — это записать два файла без заголовков на диск. Если у вас есть программа PCopier или PCopier Plus, то задача несколько упростится. Достаточно будет

---

* Прошу извинить, что я часто привожу числа в десятичной системе счисления: сказывается застарелая привычка.

скопировать файлы на диск, а затем выполнить следующую программу:

```
10 RANDOMIZE USR 15619: REM :
   LOAD "less 001"CODE 16384
20 RANDOMIZE USR 15619: REM :
   SAVE "screen"SCREEN$
30 RANDOMIZE USR 15619: REM :
   LOAD "less 002"CODE 25000
40 RANDOMIZE USR 15619: REM :
   SAVE "main"CODE 25000,32768
```

Разумеется, адреса и имена файлов пригодны только для данного примера.

Существует, однако, немало способов копирования файлов и без использования специальных копировщиков. Приведу наиболее простой для данного примера метод (в принципе, он пригоден для большинства случаев).

После загрузки файлов загрузчик, как правило, запускает программу. Обычно для этого используется уже упомянутая команда JP XXXXX, где XXXXX — адрес запуска программы. Чтобы получить копию файла на диске, нужно загрузить файлы в память с ленты и записать их на диск. Для этого можно воспользоваться уже готовым загрузчиком, предварительно заменив инструкцию, запускающую программу, на инструкцию, возвращающую управление бейсик-интерпретатору, то есть вместо мнемоники JP нужно поставить RET, код которой — 201. В нашем примере этого можно добиться, выполнив оператор POKE 65026,201. Значение по адресу, где расположена инструкция JP, изменится на 201 (команда RET), что и обеспечит возврат в бейсик-систему после загрузки обоих файлов.

Подготовив кодовый загрузчик подобным образом, можно его запускать, не забыв обеспечить запись файлов на диск непосредственно после завершения работы загрузчика. Удобно набрать программу в несколько строк:

```
10 RANDOMIZE USR 65000: REM Запуск загрузчика
20 RANDOMIZE USR 15619: REM : SAVE "screen"SCREEN$
30 RANDOMIZE USR 15619: REM :
   SAVE "main"CODE 25000,32768
```

Когда на экране появится сообщение об успешном выполнении программы, Вам останется только написать загрузчик с диска, с чем, я надеюсь, трудностей не возникнет.

Глава 5

КАК БОРОТЬСЯ С RAMTOP'ом

Часто спрашивают: «Как переделать программу на диск, если граница RAMTOP установлена слишком низко и не позволяет нормально работать с системой TR-DOS?». Попытаюсь ответить на этот вопрос.

В одной из предыдущих глав предлагался способ экономии памяти посредством разбиения загрузчиков на несколько частей. Однако такой способ является далеко не самым изящным, да и помогает не всегда. Сложности начинаются уже тогда, когда RAMTOP опускается до адреса 24500. На всякий случай напомню, что RAMTOP — это не фамилия, это адрес последней ячейки памяти, используемой Бейсиком. Устанавливается RAMTOP оператором CLEAR <адрес>

При попытке переделать на диск программу с наипростейшим загрузчиком

```
10 BORDER 1: PAPER 1: INK 1: CLS
20 CLEAR 24199
30 LOAD ""CODE
40 RANDOMIZE USR 49152
```

проблемы начинаются уже в тот момент, когда Вы успешно вывалились в Бейсик по **Break** и пытаетесь выйти в TR-DOS с помощью оператора RANDOMIZE USR 15616. Лаконичное сообщение **Out of memory** не упустит шанса появиться на экране. Вероятно, после нескольких попыток Вы решите, что эту программу гораздо приятней загружать с кассеты или «скинуть» @безьяньей кнопкой.

Мне рассказывали, как горе-программисты меняют CLEAR 24000 на CLEAR 25000, записывают программу таким образом, и она у них даже как-то работает. Я же Вам скажу следующее: этого делать нельзя ни в коем случае. Если Вы полтора миллиона раз подряд сбросите программу @безьяной, то, скорее всего, навредите ей меньше, чем подобной заменой.

Давайте лучше подумаем, как решить эту задачу более изящным способом. Вы не хуже меня понимаете, что если системе TR-DOS не хватает рабочего места, это место нужно освободить. А именно, необходимо поднять RAMTOP примерно до адреса 24700 или выше, то есть спрятать куда-нибудь примерно 500–700 байт. Здесь, в зависимости от длины и расположения файлов программы, могут возникнуть две ситуации. Первая — кодовый файл программы достаточно короткий,

вторая — кодовый файл длинный и занимает всю или почти всю память, начиная от RAMTOP и до самого верха (до адреса 65535).

Сначала рассмотрим более простой — первый случай.

Предположим, что кодовый файл, для которого необходимо написать загрузчик с диска, имеет адрес начала 24200 и длину 40800, а загружается с ленты и запускается приведенным выше загрузчиком. Если подсчитать, то получится, что адрес последнего занимаемого файлом байта равен 64999, а 536 байт с адреса 65000 фактически остаются свободными*. Итак, установив этот факт, Вы обзовете идиотом того, кто поместил файл именно в эти адреса, в результате чего 536 байт вверху пропадают, а внизу так тесно, что ощущаешь себя пассажиром троллейбуса в час пик (или скорее потенциальным пассажиром, безуспешно пытающимся залезть в переполненный троллейбус). Надо заметить, что обзывать никого не стоит, поскольку у автора программы могли быть веские причины на то, чтобы поместить ее именно туда, куда он ее поместил. Кроме того, автор, вероятно, не предполагал, что кому-то приспичит переделывать его программу на диск да к тому же в системе TR-DOS.

Первое, что приходит на ум, — это загрузить файл чуть выше (в данном случае логично использовать все 536 байт), а затем, когда обращений к диску уже не будет (то есть системе теперь не понадобится память под буферы и т. п.), поставить файл на место и запустить. Все, казалось бы, проще простого, и не исключено, что Вы поспешили написать на Бейсике нечто вроде

```
FOR n=24736 TO 65535: POKE n-536,PEEK n: NEXT n
```

Если у Вас хватило терпения дождаться результатов работы этой маленькой, но гаденькой программки, то Вы, вероятно, убедились, что она честно выполняет свое дело, но запускать ее нужно вечером, а утром, почистив зубы, можно будет поиграть в игрушку, которая наконец заняла свое привычное место и запустилась.

Я не предлагаю отказаться от этого метода, поскольку он действительно наиболее прост и удобен, я предлагаю небольшую подпрограмму на ассемблере, которая поможет переместить нужный кусок памяти на другое

* Если не верите — можете посчитать на пальцах: кстати, Вы зря смеетесь: если использовать двоичную систему счисления, то на пальцах без особых ухищрений можно показать число в диапазоне от 0 до 1023, а если прибавить еще шесть спичек, то вся память Speccy будет у Вас как на ладони.

место за несколько более короткое время:

```
,33,SL,SH     LD HL,SOURCE
17,TL,TH      LD DE,TARGET
1,LL,LH       LD BC,LENGTH
195,195,51    JP 13251
```

– где SOURCE — адрес настоящего местоположения файла; TARGET — адрес желаемого местоположения файла; LENGTH — его длина. В нашем случае значения будут равны, соответственно, 24736, 24200 и 40800.

Если Вы не знаете толком, что такое ассемблер и кому в голову взбрело его изобрести, то Вас больше заинтересует левое поле. Это коды приведенных мнемоник в десятичной системе счисления. Загадочные SL, SH, TL, TH, LL и LH означают старшие (H) и младшие (L) байты двухбайтовых чисел. Как известно, байт может принимать значения от 0 до 255, поэтому для размещения в памяти числа 24736 необходимы как минимум два байта. Младший байт для SOURCE вычисляется как

```
LET SL=SOURCE-256*INT (SOURCE/256)
```

– старший как

```
LET SH=INT (SOURCE/256)
```

Остальные числа рассчитываются аналогичным способом. Для получения значений байтов можно применить и другой, более интересный приемчик:

```
RANDOMIZE SOURCE: LET SL=PEEK 23670:
   LET SH=PEEK 23671
```

Итак, скрупулезно разложив все исходные данные на отдельные байты, Вы получили некий набор чисел. Для уверенности можете проверить правильность проведенных операций:

```
LET SOURCE=SL+256*SH
```

Набор чисел в нашем примере будет выглядеть так:

```
33,160,96
17,136,94
1,96,159 195,195,51
```

Теперь Вы спросите, что же с ними делать. Числа можно приписать к началу Вашего файла (он, таким образом, станет на 12 байт длин-

нее) и сохранить его в новом виде. Ввести числа в память можно, непосредственно выполняя операторы POKE один за другим:

```
POKE 24724,33
POKE 24725,160
POKE 24726,96
```

и т. д., либо написав несколько строк на Бейсике:

```
10 FOR n=24724 TO 24735
20 READ s
30 POKE n,s
40 NEXT n
50 DATA 33,160,96,17,196,94,1,96,159,195,195,51
```

Имеется в виду, что файл уже загружен в память на 536 байт выше (например, оператором LOAD "name"CODE 24736).

Подготовив файл таким образом, запишите его на диск:

```
SAVE "name"CODE 24724,40812
```

и напишите загрузчик с диска:

```
10 BORDER 1: PAPER 1: INK 1
20 CLEAR 24723
30 RANDOMIZE USR 15619: REM : LOAD "name"CODE
40 CLEAR 24199
50 RANDOMIZE USR 24724
60 RANDOMIZE USR 49152
```

Обратите внимание, что первый CLEAR устанавливает RAMTOP на единицу меньше, чем адрес загрузки файла, затем, после загрузки файла, устанавливается «оригинальный» RAMTOP и выполняется написанный нами фрагмент в машинных кодах (строка 50), который и помещает файл на нужное место. Затем программа нормально запускается (строка 60).

Если в той программе, которую Вы захотите адаптировать, адрес загрузки файла примерно такой же, как в приведенном примере, то проблем возникать не должно. Однако нередки ситуации, когда граница RAMTOP, которую устанавливает загрузчик, настолько низка, что при попытке выполнения второго оператора CLEAR, Вы получите уже знакомое сообщение об ошибке: 4 Out of memory, 40:1. Такая ситуация может возникнуть в связи с тем, что, как уже говорилось, TR-DOS резервирует под свои системные переменные 112 байт.

С этим бороться несколько сложнее, однако тоже можно. Ниже приводится программа в кодах, которая поможет решить эту проблему. Необходимо заметить, что работать она будет только в том случае,

если адаптируемая программа запускается единожды и в Бейсик больше не возвращается.

```
49,CL,CH      LD      SP,STACK
33,EL,EH      LD      HL,ENTRY
229           PUSH    HL
33,SL,SH      LD      HL,SOURCE
17,TL,TH      LD      DE,TARGET
1,LL,LH       LD      BC,LENGTH
195,195,51    JP      13251
```

– где STACK — адрес размещения машинного стека (для решаемых нами задач он должен совпадать с параметром оператора CLEAR); ENTRY — входная точка программы (то есть число, которое следовало за оператором RANDOMIZE USR в исходном загрузчике). В нашем примере STACK будет равен 24199, а ENTRY — 49152.

Если эту подпрограмму добавить к основному файлу так же, как я описывал, то загрузчик на Бейсике будет выглядеть следующим образом:

```
10 BORDER 1: PAPER 1: INK 1
20 CLEAR 24716
30 RANDOMIZE USR 15619: REM : LOAD "name"CODE
40 RANDOMIZE USR 24716
```

Обратите внимание, что в новом загрузчике, в отличие от предыдущего, нет ни оператора CLEAR 24199, ни RANDOMIZE USR 49152, эти операции выполняются приведенной подпрограммой в машинных кодах.

Сразу хочу предупредить, что вполне возможно возникновение различных нюансов, и Ваша программа все-таки не заработает. Это может произойти из-за того, что программа после выполнения каких-либо операций все же возвращается в Бейсик и продолжает в нем работать. Это может привести к самым разнообразным эффектам, однако подобные ситуации встречаются достаточно редко.

С этими довольно простыми случаями Вы, вероятно, уже разобрались (иначе — читайте все с начала; если и это не поможет — обзовите автора неприличным словом как несостоявшегося).

Теперь разберем более сложную и неприятную ситуацию, когда, после тщательных расчетов (быть может, на пальцах), Вы пришли к прискорбному выводу, что файл занимает всю память от RAMTOP и выше, а так необходимые 500–700 байт взять совершенно негде. Не пугайтесь, это еще не повод для беспокойства. Вы забыли о довольно большом резерве — видеопамяти и буфере принтера. Использование буфера принтера предпочтительнее, но он имеет объем лишь 256 байт (с адреса 23296), поэтому может помочь только, если недостает каких-нибудь 200–250 байт.

По большому счету, в случае длинного файла (имеется в виду файл, занимающий все пространство от RAMTOP и выше) организация загрузки и запуска будет во многом аналогична организации загрузки и запуска короткого файла. Единственным принципиальным отличием будет то, что длинный файл придется разбить на два коротких. Способов разбиения файлов на куски существует достаточно много. Это можно сделать и непосредственно на диске, с помощью программы Disk Doctor*.

Подробнее рассмотрим ситуацию, когда исходный файл находится еще на кассете. Самое простое — загрузить файл в память, не забыв установить правильный (оригинальный) RAMTOP, и записать два файла на кассету оператором SAVE. При этом имейте в виду, что выходить в TR-DOS ни до, ни после загрузки файла ни в коем случае нельзя.

Предположим, Вы захотели адаптировать программу со следующими исходными данными:

загрузчик:

```
10 CLEAR 23999: LOAD ""CODE
20 RANDOMIZE USR 24000
```

файл: адрес загрузки — 24000, длина — 41536.

Сбросьте машину (аккуратно, так, чтобы телевизионный кабель не оборвался), затем проделайте следующие операции:

```
CLEAR 23999
LOAD ""CODE
```

— загрузите файл, далее выполните

```
SAVE "1"CODE 24000,1000
SAVE "2"CODE 25000,40536
```

Итак, на кассете Вы получили два файла. Вновь сбросьте машину и загрузите первый (более короткий) файл в область экрана оператором

```
LOAD "1"CODE 16384
```

— затем запишите его на диск:

```
RANDOMIZE USR 15619: REM : SAVE "1"CODE 16384,1000
```

Второй файл перенесите на диск любым доступным способом.

---

* Как автор этой программы рекомендую пользоваться версиями с номерами не менее 3.2. В настоящий момент последняя версия имеет номер 4.3. Программой же DISK DOCTOR фирмы Technology Research, которая поставлялась в комплекте с системой TR-DOS, пользоваться можно только после принятия некоторой дозы алкоголя.

Можно несколько ускорить операцию разбиения файла, если после записи на кассету первого куска (длиной 1000 байт) выполнить оператор CLEAR 24999 (для данного примера, разумеется). Таким образом, мы освобождаем место для работы TR-DOS, поэтому второй файл можно будет записать на диск непосредственно из памяти, избежав промежуточных стадий. Обращаю Ваше внимание на то, что после выполнения оператора CLEAR 24999 область памяти с этого адреса и ниже отводится под Бейсик, и тем самым портится кусок программы, но поскольку мы только что этот кусок сохранили на кассете, бояться абсолютно нечего.

Итак, мы имеем на диске два файла, и если просмотрим каталог оператором LIST системы TR-DOS, то увидим, что адрес загрузки первого файла 16384 и длина 1000, второго — 25000 и 40536, соответственно (чего, в общем-то, и следовало ожидать). С этими файлами необходимо поступать следующим образом: первым, без всяких хитростей (RAMTOP должен быть установлен ниже адреса загрузки файла), загружается файл "2", а лишь потом — "1". Поскольку загружаться файл "1" будет в экранную область, перед его загрузкой, из эстетических соображений, рекомендуется сделать цвет INK и PAPER одинаковыми и очистить экран. Если в программе есть заставка, то она должна загружаться первой, затем длинный файл, и после очистки экрана — короткий.

Чтобы загруженный в экран файл поместить на его родное место и запустить программу, можно воспользоваться подпрограммой в машинных кодах, приведенной на стр. 25. В разбираемом примере эта подпрограмма будет выглядеть следующим образом:

```
LD    HL,24000
PUSH  HL
LD    SP,23999
LD    HL,16384
LD    DE,24000
LD    BC,1000
JP    13251
```

Опять-таки имейте в виду, что адаптированная программа будет работать только в том случае, если она не возвращается в Бейсик.

Во всех приведенных в данной главе примерах рассматривались программы с одним кодовым файлом. Все предлагаемые решения можно применить и к программам с несколькими файлами. Иногда

могут возникнуть сложности, однако я уверен: немного попрактиковавшись, Вы настолько хорошо освоите подобные приемы, что уже мне у Вас придется учиться, что и как адаптировать на диск.

Глава 6

ОРГАНИЗАЦИЯ ЗАГРУЗЧИКОВ В МАШИННЫХ КОДАХ

В этой главе я расскажу, как на практике можно организовать кодовый загрузчик. Приемы, описанные здесь, можно использовать также для внедрения в Ваши бейсик-программы небольших подпрограмм в машинных кодах.

Вообще говоря, об адаптации программ на диск в этой главе говориться не будет, однако у человека, способного перемешать Бейсик с кодом, поубавится проблем с адаптацией.

Самый простой, понятный и системный, но глупый и неинтересный способ организации загрузчика в машинных кодах был описан в четвертой главе (внесу ясность: под организацией я подразумеваю то, как загрузчик загружается и запускается, а не то, как он работает). На практике встречи с таким загрузчиком вызывают изумление, настолько они редки. Способов организации загрузчиков так много, что использовать «самый простой, понятный и системный» приходит в голову редким индивидуумам.

Наиболее распространенным методом организации кодового загрузчика является загрузка его вместе с бейсик-программой. Попробую объяснить, как можно перемешать Бейсик с машинным кодом, чтобы подобный симбиоз работал так, как хотелось бы.

В первом приближении можно выделить три способа внедрения машинного кода в Бейсик:

1. в виде комментария;
2. в виде программы на Бейсике;
3. в виде переменных.

Эти фразы могут вызвать законное удивление и ряд недоброжелательных эпитетов в мой адрес, поэтому попробую реабилитироваться и объяснить, что они значат.

Первый способ заключается в следующем: загрузчик помещается на место комментария, следующего за оператором **REM**. Это не значит, что требуется долго и мучительно изобретать комментарий на языках народов Крайнего Севера. Можно просто набить необходимое количество пробелов. Если Вы так же тщеславны, как и я, то можете набивать свои паспортные данные — так или иначе, вся

введенная информация будет впоследствии замещена машинными кодами.

Если Ваши загрузчики или подпрограммы в машинных кодах достаточно велики, то способы выделения места под них должны отличаться от вышеприведенного. Ведь если на бумаге написать комментарий, состоящий из трех тысяч пробелов, достаточно просто (как впрочем и из любого другого количества пробелов), то для того, чтобы ввести этот текст с бумаги в машину, необходимо нанимать опытную машинистку.

Мне кажется, что человек, способный написать загрузчик длиной в несколько килобайт, и без моих наставлений сможет грамотно выделить место для своей программы. Тем не менее, могу посоветовать воспользоваться для этого одной из программ пакета Supercode либо функцией CREATE системного монитора Mon2 (не путайте с MONS), если он у Вас есть, и Вы умеете им пользоваться.

Второй способ практически не отличается от первого, разница лишь в том, что программа в машинном коде занимает не поле комментария, а поле операторов, то есть отсутствует оператор REM. Поскольку интерпретатору абсолютно безразлично, что находится в строке вслед за REM, то первым способом можно располагать машинный код в произвольном месте программы. При использовании второго способа необходимо проследить все возможные «ходы» интерпретатора: нельзя допускать, чтобы в процессе работы интерпретатор «набрел» на строку, в которой находится машинный код, иначе его может стошнить.

Третий способ более сложен и интересен. Машинный код записывается в область переменных Бейсика. Этого можно добиться, например, задав массив и записав в зарезервированную под него область памяти свою программу в кодах.

Вообще говоря, все три способа очень похожи друг на друга, разница состоит в том, каким образом рассчитать адреса, по которым будет размещаться машинный код. Если Вы используете первый или второй способ, то машинный код можно разместить по фиксированным адресам, записав его в первую строку бейсик-программы (не лишним будет придать этой строке нулевой номер). В этом случае Вы сможете практически беспрепятственно редактировать бейсик-программу: адрес запуска программы в машинных кодах не изменится. Он будет равен 23760 до инициализации TR-DOS и 23872 после инициализации. Откуда взялись эти значения, легко подсчитать. Вы уже знаете, что в обычных условиях программа на Бейсике начинается с адреса 23755. Байты, расположенные по этому и последующим адресам, определяют номер строки (23755 и 23756), длину строки (23757 и 23758), затем следует поле операторов. В нашем случае — это оператор REM (адрес 23759), после которого, начиная с адреса 23760, следует комментарий (если Вы уже поместили туда какую-нибудь гадость, то комментарий может быть весьма трудночитаемым).

Если же Вы захотите поместить программу в машинных кодах где-нибудь в середине или в конце бейсик-программы, то рассчитать адрес ее расположения будет несколько сложнее. Для этого придется вычислить адрес начала строки, в которой программа записана, и к нему прибавить 5. Дополнительным недостатком подобного способа

размещения кодов является то, что после редактирования или добавления в бейсик-программу строк с номерами меньшими, чем номер строки с машинным кодом, адрес начала машинного кода будет смещаться. То же самое будет происходить при размещении машинного кода в области переменных.

Третий способ наиболее трудоемок, и пользоваться им рекомендуется только тогда, когда нужно посильнее запудрить мозги тому, кто захочет в этой программе разобраться. Адрес машинного кода, размещенного в области переменных, рассчитать сложней, чем в уже рассмотренных случаях, так как необходимо знать распределение памяти, выделенной для переменных Бейсика. До сих пор мне встречался только следующий способ: машинный код размещался, начиная с самой первой ячейки, отводимой под переменные. Ее адрес можно определить оператором

```
PRINT PEEK 23627+256*PEEK 23628
```

Если машинный код размещается подобным образом, то использование каких-либо переменных в бейсик-программе полностью исключается. Вы можете выполнить последовательность операторов

```
LET a=100: PRINT a
```

и, если не вылетите куда-нибудь, то, скорее всего, получите сообщение

```
2 Variable not found, 0:2
```

При использовании переменных бейсик-программы для размещения машинного кода возникает один интересный эффект: такую бейсик-программу, записанную с автостартом на кассету или диск, невозможно загрузить оператором MERGE.

Для того чтобы программа не загружалась оператором MERGE, можно также посоветовать указать неправильную длину одной или нескольких строк бейсик-программы. Это, во-первых, помешает их редактировать, а во-вторых, оператор MERGE при загрузке, после некоторых раздумий, покажет загружающему фигу. Сбросьте компьютер и наберите любую программку из одной строчки, а затем выполните оператор POKE 23757,255 и попробуйте отредактировать эту строчку. Эффекты могут быть самыми неожиданными.

Если Вы все же решили использовать переменные для хранения программы в машинных кодах, то выделить для этого область можно следующим образом:

```
CLEAR: DIM a$(x)
```

— где x — длина Вашей программы в байтах (на самом деле памяти выделяется несколько больше).

Оператор CLEAR очищает область переменных. Это необходимо для того, чтобы массив a$ размещался в самом начале этой области, и можно было легко вычислить адрес его начала.

Размещая программу в машинных кодах в области переменных, будьте осторожны. К каким эффектам при этом может привести выполнение операторов CLEAR или RUN, не знают ни сэр Клайв Синклер, ни доктор Логан ни, тем более, я. Определенно сказать можно только то, что программа работать не будет. Что касается меня, то я ни разу не использовал область переменных в таких целях и, тем не менее, не чувствую себя в чем-то ущербным.

В предыдущей главе Вашему вниманию предлагались подпрограммы в машинных кодах, помогающие быстро перемещать большие области памяти и запускать программы. Значительно удобнее организовывать эти подпрограммы не так, как было предложено, то есть дописывая их к основному файлу, а внедрять их в Бейсик одним из приведенных способов.

Для примера внедрим в Бейсик программу в кодах, приведенную на стр. 23. В загрузчик на Бейсике необходимо первой строкой вставить оператор REM и вслед за ним «комментарий», состоящий из 12 пробелов или других символов. Затем нужно заменить «комментарий» на машинный код с помощью оператора POKE (система TR-DOS должна быть инициализирована):

```
POKE 23872,33
POKE 23873,160
POKE 23874,96
POKE 23875,17
```

и т. д., до полного удовлетворения.

Адрес запуска этой подпрограммы будет равен адресу ее начала, то есть 23872. В результате Ваших действий новый вариант загрузчика будет выглядеть так:

```
 5 REM <Здесь будет всякая гадость. Возможно даже,
   что Вы не увидите 10-й строки, в этом случае
   выполните LIST 10>
10 BORDER 1: PAPER 1: INK 1
20 CLEAR 24735: REM Обратите внимание!!!
30 RANDOMIZE USR 15619: REM : LOAD "name"CODE
40 CLEAR 24199
50 RANDOMIZE USR 23872
60 RANDOMIZE USR 49152
```

Таким образом Вы избавляетесь от необходимости дописывать к файлу какие-либо куски и оставляете за собой возможность не трогать файл вообще, а загружать его в память оператором LOAD "name"CODE 24736 системы TR-DOS.

Я полагаю, сейчас Вам стало ясно, что оператор POKE — хоть и универсальное средство для программирования в машинном коде, но его применение — довольно занудное занятие, а результат работы

плохо поддается проверке. Если Вы где-то ошиблись, то найти ошибку будет довольно сложно, даже если использовать функцию PEEK. Так что мой Вам совет: немедленно бегите к друзьям, спекулянтам, к кому угодно; просите, покупайте, воруйте (воровать нехорошо!) описания языка ассемблера, программ ассемблера и отладчика*; садитесь к машине и изучайте все это. Если я когда-нибудь напишу продолжение этой книжки, то наверняка буду подразумевать, что Вы уже имеете хоть какое-то представление об ассемблере.

Глава 7

КСОРКА

(обещанное мной продолжение, написанное специально для этого издания)

Это странное, даже неприятное слово было придумано мною несколько лет назад и, говорят, прижилось. В этой главе я попытаюсь объяснить, что оно значит, и если Вы поймете, какой страшный смысл оно таит, то свою задачу я выполню.

Итак, поехали. В предыдущих главах уже приводились кое-какие элементарные загрузчики в кодах. Но, как и простые бейсик-загрузчики, легкие защиты в кодах встречаются не так часто, как хотелось бы. Все, кому не лень, пытаются более или менее успешно закрыть свои загрузчики от посторонних глаз, и эти ухищрения создают определенные сложности при переделке программ на диск.

Рассмотрим простенькую программку, которая, будучи запущена, оставит от вполне связного текста загрузчика нечто совершенно неудобоваримое:

```
        ...
        LD      HL,START    ; адрес начала загрузчика
        LD      BC,LENGTH   ; длина загрузчика
LOOP    LD      A,(HL)      ; загрузка в A из памяти
        XOR     176         ; изменение байта в A
        LD      (HL),A      ; запись его обратно
        INC     HL          ; увеличение указателя
        DEC     BC          ; увеличение счетчика
        LD      A,B         ; счетчик
        OR      C           ;   равен 0?
        JR      NZ,LOOP     ; если нет, то на LOOP
        ...
```

* Для начала можно посоветовать книгу [1], где подробно описан пакет DEVPAC — программы GENS4 и MONS4.

Предположим, что с адреса START у нас находится загрузчик длиной LENGTH байт. Если разобраться, как работает приведенный фрагмент, то очевидно, что после его выполнения никакой дизассемблер не поможет нам прочитать содержимое загрузчика до тех пор, пока мы не выполним этот фрагмент еще раз.

Полагаю, что уже кое-что становится понятным. В частности, ясно, куда пропадают загрузчики — да никуда, просто их авторы вставляют в них куски вроде приведенного. Когда же программа стартует, она сначала «расксоривается», а потом запускается. Возможно, уже стало понятно происхождение слов «ксорка», «заксорен», «расксоривается» и т. п. Дело вот в чем: излюбленной командой для заксоривания является XOR, так как ее повторное выполнение (с тем же параметром) приводит к тому, что операнду, с которым она работает, возвращается прежнее значение (которое он имел до первого выполнения XOR). Без этой команды, похоже, не обходится ни одна более или менее приличная защита.

Как же бороться с этим наваждением? Тут, пожалуй, можно дать, разве что, несколько теоретических советов, не более, — фантазия у людей весьма богатая, а когда эти люди еще и программисты...

Итак, способ первый. Он наиболее прост, но часто оказывается самым эффективным. Для его применения Вам нужно, естественно, хорошо разобраться с каким-нибудь монитором-отладчиком. Из ранее упомянутых отладчиков наиболее удобным для хакерских дел я считаю Mon2, а MONS4 более пригоден для отладки программ. Оба они имеют режимы трассировки, и часто бывает достаточно найти входную точку загрузчика, точнее — ксорки, и протрассировать ксорку от начала до конца. В результате Вы получите вполне приличный листинг загрузчика и, полагаю, еще раз перечитаете предыдущие главы этой брошюры. Не стоит бояться двух, пяти, ста вложенных ксорок (это не шутка, защита типа Alcatraz protection, например, содержит их около трехсот!). Если в ксорках не используются какие-нибудь специальные «извраты», то они могут быть с легкостью раскручены.

Но, вероятно, Вы уже догадываетесь, что не все так просто в этой жизни. Есть масса ухищрений, которые мешают трассировать программы из отладчиков. Основные попробую перечислить.

1. Использование недокументированных команд. Как известно, в Z80 есть некоторое количество команд, не описанных ни в каких фирменных справочниках. Даже если Вы сами хорошо представляете,

что эти команды означают, то отладчику от этого легче не становится, и он либо «затыкается», либо игнорирует их, либо еще что-нибудь придумывает, чтобы отмазаться.

2. Использование регистра R, специального регистра регенерации динамической памяти, содержимое которого увеличивается на единицу после выполнения каждой команды (точнее, очередного цикла) микропроцессора. Обычно этот регистр используется разве что для генерации случайных чисел, однако ловкие программисты придумали, как его применить. Например, так:

```
        ...
        LD      HL,START
        LD      BC,LENGTH
        XOR     A           ; обнуление
        LD      R,A         ;   регистра R
LOOP    LD      A,R         ; получение очередного значения R
        XOR     (HL)        ; заксоривание
        LD      (HL),A      ; запись заксоренного значения
        INC     HL
        DEC     BC
        LD      A,B
        OR      C
        JR      NZ,LOOP
        ...
```

В результате выполнения этого фрагмента в памяти образуется еще больший бардак, чем после ксорки, описанной в начале главы, но при повторном его выполнении все восстанавливается. Вам же мешает только одно: когда Вы трассируете этот участок отладчиком, то выполнение одной команды отлаживаемой программы сопровождается выполнением сотен или тысяч команд самого отладчика. Понятно, что в результате живущий своею жизнью регистр регенерации памяти принимает совсем не то значение, которое от него ожидали, и в памяти вместо загрузчика образуется полный хаос. Должен заметить, что это — хроническая болезнь отладчиков: для Speccy я не встречал еще ни одного, который бы правильно обращался с регистром R.

3. Использование второго режима прерываний. Этот очень экзотический способ защиты от трассировки встречается крайне редко из-за чрезвычайной сложности его реализации, так что я даже не буду на нем останавливаться.

Рано или поздно Вы столкнетесь с загрузчиком (точнее, с ксоркой), который нельзя расксорить, используя отладчик в пошаговом режиме. Что же можно предпринять? Если быть до конца откровенным, то Вам может помочь только Ваша фантазия: пусть она окажется богаче фантазии автора защиты. Однако для затравки попробую дать не-

сколько советов. Возьмем приведенную выше ксорку, несколько ее видоизменив:

```
        ...
        LD      HL,START
        LD      B,LENGTH
        XOR     A
        LD      R,A
LOOP    LD      A,R
        XOR     (HL)
        LD      (HL),A
        INC     HL
        DJNZ    LOOP
        JP      OUTTHERE
```

Здесь можно применить одно простое средство: поставить точку останова после команды перехода к циклу (DJZN). В данном случае мы можем себе это позволить, так как после команды перехода к циклу стоит команда JP. В принципе, на этом месте может стоять любая осмысленная команда, которая сама не участвует в ксорке и не заксорена.

Но посмотрите, что получится, если в нашем примере адреса будут расположены следующим образом:

```
        ...
        LD      HL,START
        LD      B,LENGTH
        XOR     A
        LD      R,A
LOOP    LD      A,R
        XOR     (HL)
        LD      (HL),A
        INC     HL
        DJNZ    LOOP
START   ...
```

Поставленная по адресу START точка останова будет испорчена уже после выполнения первых шагов ксорки, и, разумеется, ни в какой отладчик Вы уже не вернетесь. За этим необходимо следить самым тщательным образом.

После следующей фразы, возможно, меня назовут отступником, но я все же ее произнесу: от многих сложностей Вам поможет избавить-ся... кнопка Magic. Это, конечно, не значит, что я изменил своим принципам, это значит только, что кнопка Magic — отличное средство бороться с ксорками.

Дело в том, что когда Вы обычным образом запустите заксоренный загрузчик, он расксорится в памяти и начнет выполнять свои прямые обязанности, то есть загружать программу. Если в этот момент нажать

кнопку Magic, то на диск запишется файл, содержащий копию памяти. В нем, среди прочего мусора, Вы сможете отыскать расксоренный загрузчик. Запустите программу Disk Doctor (или аналогичную) и потихоньку изучайте этот файл, которым кнопка Magic загадила Ваш диск...

Тут, ребята, я не могу сказать ничего более конкретного, кроме того, что Вы будете иметь сорок восемь килобайт «нечта», из которого малопонятным путем надо будет вытянуть загрузчик. По сути, нужно решить две задачи: первую — определить, где загрузчик находится (это довольно просто) и вторую — найти его входную точку. Вот здесь могут возникнуть вопросы. Однако, если Вы тщательно проработали ксорку до того, как нажали кнопку Magic, то количество вопросов наверняка уменьшится.

Я не полный идиот и отдаю себе отчет в том, что сказанное не может претендовать на полноту, но поверьте, объяснять на бумаге то, что понимается не разумом, а воспринимается интуитивно, крайне тяжело. Поэтому эта глава не содержит конкретных советов, а, скорее, намекает на возможные решения.

Для того, чтобы Вы получили более яркое (или хотя бы какое-нибудь) представление о возможных вариантах ксорок и способах борьбы с ними, я попробую описать несколько наиболее выдающихся из известных мне.

Защита типа Alcatraz protection

Это самая навороченная защита, которую я встречал. Программы, закрытые ею (например, Infiltrator, Rygar) не копируются стандартными копировщиками, применен ускоренный формат записи на ленту и ряд других ухищрений, но нам интересно не это, а то, какие в ней применены ксорки. А их там порядка 300, причем попадаются как и самые идиотично простые, подобно приведенной в начале главы, так и ксорки с использованием недокументированных команд, регистра регенерации, стека и т. п. Самым же интересным является то, что сам загрузчик загружает заксоренный файл, причем файл содержит новый загрузчик, который загружается поверх старого, модифицируя при этом аргументы ксорки. И так три раза. В общем, тяжело для раскрытия, хотя и не особенно трудно для понимания.

Снять эту защиту честным путем мне не удалось, помогла только кнопка Magic, чего и Вам желаю.

Да, забыл сказать, что заставка в этой защите загружалась настолько красиво...

Защита типа Speedlock protection

Чрезвычайно распространенная защита (встречалась, например, в Freddy Hardest, Tanx и в десятках двух программ).

Полагаю, что примерно 70–80 процентов программ снабжались ею (речь, разумеется, идет об оригинальных фирменных программах, облаченных в красивые разноцветные упаковки и снабженных описани-

ем, а не о BillGilbert-S.S.Captain-JackO'Lantern-NicolasRodionov'вских взломах).

Защита, как впрочем и все защиты, была предназначена для ограничения распространения программ, то есть программы, закрытые ею, не копировались.

Мне попадалось несколько модификаций Speedlock'а. Наиболее талантливым, на мой взгляд, был первый вариант (разработан в 1984 году, автор, кажется, David Aubrey Jones), о нем я расскажу подробнее. Остальные варианты были разработаны уже другими авторами и вряд ли заслуживают внимания с точки зрения ксорок, хотя внешне они работали довольно забавно.

Для 1984 года в Speedlock'е применялись просто гениальные штучки. Я весьма благодарен ее автору, так как для меня эта защита была школой, пройдя которую, я понял очень многое. К сожалению, вряд ли Вам удастся столкнуться со Speedlock'ом, но если Вы захотите пройти школу «молодого бойца», то могу порекомендовать повозиться с моей программкой Disk Control Utility. Начинать лучше с версий не позже 2.12, в них использована ксорка, написанная мною году в 87, если не раньше, а она лишь жалкая пародия на Speedlock. В версиях позже 2.12 ксорка написана в 1992 году, а это уже не для «молодых бойцов», лучше попрактиковаться на чем-нибудь попроще.

Итак, более подробно о Speedlock'е. Листинг бейсик-загрузчика выглядел весьма разумно и понятно, а именно, примерно так:

```
0

                    Speedlock protection
```

После первичной обработки можно было выудить около десятка строк с нулевым номером, в каждой из которых было несколько операторов POKE. Других операторов не встречалось. Бейсик, разумеется, заканчивался мусором, то есть загрузчиком в кодах.

Операторы POKE выполняли следующее: переустанавливали системную переменную ERR_SP (адрес возврата по ошибке) так, что как только интерпретатор выполнял всю разумную часть бейсик-программы и

сталкивался с неразумной, то он тут же вылетал, но не в процедуру обработки ошибки с целью написать сообщение Nonsense in Basic, а на загрузчик в кодах, точнее, на раскрутку ксорки.

Раскрутка ксорки в Speedlock'е для неподготовленного человека выглядит довольно забавно, но и бессмысленно: байт пятьсот каких-то пересылок между регистрами, сложений, вычитаний, логических операций, манипулирования со стеком. Причем ни одного оператора, претендующего на то, чтобы завершить цикл ксорки. После этих пятисот байт непонятно чего следовало еще несколько байт непонятно чего. Эти несколько байт выглядели примерно так:

```
        ...
LOOP    LD      A,R
        XOR     (HL)
        LD      (HL),A
        LDI
        DEC     SP
        DEC     SP
        RET     PE
        DEC     SP
        DEC     SP
        RET     PO
        ...
```

и дальше какой-то бред. Как выяснилось при ближайшем рассмотрении, это, собственно, и есть тело ксорки, то есть цикла раскрутки. На первый взгляд трудно увидеть в этом цикл, однако нужно принять во внимание содержимое стека в момент попадания на адрес LOOP. На вершине стека был записан адрес LOOP (если быть более точным, то не на вершине, а на два байта ниже), затем адрес следующей ксорки (которая раскручивалась этой). Таким образом, после выполнения операций

```
        DEC     SP
        DEC     SP
```

счетчик стека устанавливался на указатель на адрес LOOP, а

```
        RET     PE
```

выполнял возврат по этому адресу в случае, если после выполнения

```
        LDI
```

регистр BC не равен нулю, то есть необходимо продолжить цикл. Затем, после расксоривания, указатель стека снова дважды уменьшается на единицу, устанавливаясь на LOOP, а команда RET PE снова

«возвращает» на LOOP. На всякий случай напомню, что команда LDI делает следующее:

```
LD      (DE),(HL)   ; такой мнемоники в Z80 нет, но
                    ;  чтобы описать LDI
                    ;  ее использовать можно, так
                    ;  как делает она именно это
INC     HL
INC     DE
DEC     BC
```

Причем, в зависимости от состояния BC (равно или не равно нулю), соответствующим образом устанавливается флаг процессора Parity, этим и объясняется использование команд RET PO и RET PE.

Когда же цикл завершается, то программа переходит ко второй группе операторов DEC SP, выполняет их и оператором RET PO «возвращается» на следующую расксорку.

Следующая расксорка выглядит примерно также, затем следует еще парочка попроще.

Вот такие дела.

Теперь попробую придумать ксорку с использованием стека. Ну, например, так:

```
        ...
        LD      SP,END      ; начало заксоренной области + 2
LOOP    LD      BC,LENGTH
        LD      A,R
        POP     HL          ; фактически: LD H,(SP-1)
                            ;   LD L,(SP-2)
                            ;   DEC SP
                            ;   DEC SP
        XOR     A,H         ; последовательная ксорка H и L
        LD      H,A
        LD      A,R
        XOR     L,A
        PUSH    HL          ; фактически: LD (SP),L
                            ;   LD (SP+1),H
                            ; ...INC SP
                            ; ...INC SP
        POP     HL          ; фактически: INC SP
                            ;   INC SP
        DEC     BC          ; дальше ясно
        DJNZ    LOOP
        ...
```

Кстати, не так плохо получилось. Дополнительные комментарии, на мой взгляд, излишни. Можно несколько изменить эту ксорку:

```
        ...
        LD      SP,END      ; начало заксоренной области + 2
LOOP    LD      BC,LENGTH
        LD      A,R
        POP     HL          ; фактически: LD H,(SP-1)
                            ;   LD L,(SP-2)
                            ;   DEC SP
                            ;   DEC SP
        XOR     A,H         ; ксорка H
        LD      H,A
        PUSH    HL          ; фактически: LD (SP),L
                            ;   LD (SP+1),H
                            ; ...INC SP
                            ; ...INC SP
        INC     SP          ; увеличение счетчика стека на 1
        DEC     BC          ; дальше ясно
        DJNZ    LOOP
        ...
```

Этот вариант будет работать несколько медленнее (впрочем, это вряд ли имеет принципиальное значение) и менее извращен: за каждый цикл ксорится один байт, а в первом случае за каждый цикл ксорится два байта и используются разные смещения регистра R.

Теперь несколько слов об использовании недокументированных команд. Довольно полное их описание можно найти в книге [1]. Я же остановлюсь на описании наиболее распространенных из них, а именно командах, работающих с частями индексных регистров IX и IY как с 8-разрядными регистрами общего назначения. На самом деле все не так страшно, и, столкнувшись впервые с такими командами в Speedlock'е, я понял, где собака зарыта, затратив совсем немного умственных усилий.

Как известно, доступ к регистрам IX и IY осуществляется теми же кодами операций, что и к регистру HL, за одним лишь исключением — команды работы с регистрами IX и IY префиксируются байтами 221 (DDh) и 253 (FDh) соответственно. И если в фирменных руководствах есть описания только таких команд как LD IY,0, то это еще не значит, что нет команды LD IYh,0*.

---

* Так как запись команды LD I,0 неоднозначна, части индексных регистров обозначаются IXh и IXl (старший и младший байты регистра IX) и IYh, IYl (старший и младший байты регистра IY).

Документированную команду LD IY,0 можно представить следующим образом:

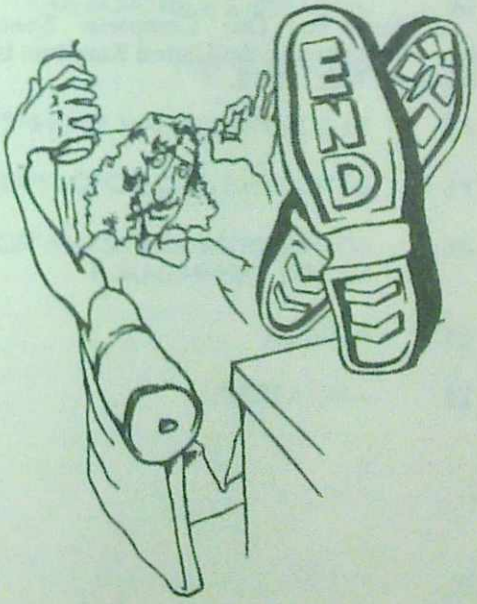
```
DEFB   253          ; префикс регистра IY
LD     HL,0
```

По аналогии с командой LD IY,0 недокументированную команду LD IYh,0 можно записать в таком виде:

```
DEFB   253
LD     H,0
```

Ассемблеры (например, GENS4), как и отладчики, недокументированные команды не понимают, но используя в них приведенную запись команды LD IYh,0, можно внедрять недокументированные команды в программы.

Ну вот пока и все. Мне остается только надеяться, что изложенное в этой брошюре будет тем толчком, который приведет Вас к истинным высотам; что очень быстро, адаптировав сотню-другую программ, Вы приобретете бесценный опыт, и начнете сами писать оригинальные программы и придумывать к ним красивые защиты.



ЛИТЕРАТУРА

1. Ларченко А. А., Родионов Н. Ю. ZX Spectrum для пользователей и программистов — СПб.: Импакс, 1991.
2. Диалекты Бейсика для ZX Spectrum — СПб.: Питер, 1992.
3. Рафикузаман М. Микропроцессоры и машинное проектирование микропроцессорных систем: в 2-х кн. Кн. 1. Пер. с англ. — М.: Мир, 1988.
4. Baeurich, H. Barthold, H.: 8-bit-Mikrorechentechnik; Prozessoren, Schaltkreise und ihre Programmierung — Berlin: Militaerverlag der Deutshen Demokratischen Republik, 1988.
5. Logan I., O'Hara F.: The Complete Spectrum ROM Disassembly — Published in the United Kingdom by Melbourne House (Publishers) Ltd., 1983.