

ТАЙНИКИ ZX-SPECTRUM

А Н Н О Т А Ц И Я

Мало кому нравятся программы, которые при первой же ошибке в ответе, либо при нажатии клавиши break очищают всю память компьютера, не оставляя после себя никакого следа, либо "зависают", вынуждая нажимать reset. Ситуация перестает быть забавной, когда мы имеем какую-нибудь программу, которую хотим приспособить для нетипового оборудования (джойстика), или хотим заменить в программе все английские тексты на русские, а программу не удается остановить.

В этом описании будет последовательно дана информация, необходимая при такого рода работе, о твоём компьютере, такая как карта памяти, способ записи в памяти отдельных строк бейсика, важные системные переменные, и т.п. Позже мы доберемся до считывания программ и блоков данных с ленты "безопасным" способом, т.е. так, чтобы они не стартовали автоматически и можно бы рассмотреть их содержание, в конце мы займемся также обезвреживанием предохранителей, написанных на внутреннем языке. Постараемся все это иллюстрировать конкретными примерами на известных программах. Надеемся, что наши усилия не пропадут зря, и ты тоже научишься лобираться без препятствий

1. Распределение памяти.

В принципе память распределена на две основные части: rom и ram. rom занимает адреса 0...16383, ram - адреса 16384...65535. Содержимым rom мы сейчас заниматься не будем, но зато внимательно присмотримся к памяти ram. Она поделена на блоки, выполняющие различные функции в системе бейсика (рис.1)

	0
ROM	16384
DISPLAY FILE	22528
ATTR	23296
буфер принтера	23552
системные переменные	23734
карта микродрайва	CHANS (23631) (23734)
информация о каналах	
+80	PROG (23635) (23755)
область системы бейсика	RAMTOR (23730) (65368)
+3E	UDG (23675) (63368)
графика определенная пользователем	R-RAMT (23732) (65535)

Рис. I Распределение памяти.

DISPLAY FILE

-область, в которой хранится информация о том включены или погашены последовательные точки экрана, это занимает 6144 байта.

ATTR

-эти 768 байт (с адреса 22528) определяют цвета последовательных полей экрана (8x8 точек).

Буфер принтера

-эта область (от 23296 до 23528) используется лишь во время работы компьютера с принтером. Если не используются инструкции, касающиеся работы компьютера с принтером (

Системные переменные

-Эти ячейки памяти используются системой для запоминания необходимых для его безошибочной работы данные, такие как например, адреса так называемых подвижных блоков памяти, информация о выполнении программы на бейсике, т.е. какая строка выполняется, к которой должен осуществиться переход, появились ли какие-либо ошибки, и т.п. В этой области находятся так же переменные, содержащие код последней нажатой клавиши, продолжительность "веер" и еще много других. Подробнее мы займемся ими позднее.

Непосредственно за системными переменными, которые кончаются адресом 23733, начинаются, так называемые, подвижные области. Это означает, что адреса их начал и концов (а также длина) могут изменяться в зависимости от того подключены ли какие-либо внешние устройства, какова длина программы на бейсике, сколько образуется переменных и т.д. Адреса подвижных блоков находятся в соответствующих системных переменных.

Адрес

На рис.1 и рис.2 за стрелкой число означает адрес начала указываемого блока. Если адрес этот перемещается, то вместо числа имя системной переменной, содержащей этот адрес, а в скобках - адрес этой переменной. В квадратных скобках находится значение этой переменной, которое устанавливается после включения компьютера (или выполнения reset), но без подключения внешних устройств.

Карта микродрайва

-если к твоему компьютеру подключен zx интерфейс I, то с адреса 23734 до адреса на 1 меньше, чем содержимое переменной `smans`, находится карта микродрайва - область, используемая как буфер для трансляции данных, как набор добавочных системных переменных и т.д. Если интерфейс не подключен, то

область эта попросту не существует - переменная `CHANS` содержит адрес 23734. Она определяет начало блока памяти, в котором содержатся данные о существующих каналах.

Информация о каналах

- необходима для безошибочного выполнения инструкций `PRINT`, `LIST`, `INPUT` и им подобных. В последней ячейке этой области находится число *80 (128), определяющее конец этого блока (это, так называемый, указатель конца).

Область бейсика

- область памяти содержит текст введенной программы на бейсике. Адрес ее начала хранится в переменной `BASE`. Сразу за текстом программы (с адреса, указываемого переменной `VARS`), находится область, в которой интерпретатор размещает переменные, создаваемые программой. Она заканчивается указателем конца. Затем начиная с адреса, содержащего в переменной `E- LINE`, находится область, используемая во время редактирования строки бейсика, а также ввода директив (т.е. когда в нижней части экрана мигает курсор и мы вводим инструкцию на бейсике). В конце этой области находятся два байта с содержимым: 13 ("ENTER") и 128 (конец этой области). Сразу после, начиная с адреса, указываемого системной переменной `WORKER`, находится подобная область, но предназначенная для ввода данных во время выполнения интерпретатором инструкции `INPUT` (завершенной знаком "ENTER"). За буфером `INPUT` (который автоматически удаляется после выполнения этой инструкции) находится

"Текущее рабочее пространство" - место памяти, используемое для самых различных целей. Туда, между прочим загружаются заголовки считанных с ленты программ, туда считывается программа, размещаемая в памяти с помощью `MERGE`, прежде чем будет присоединена к уже существующей программе. Эта область используется тогда, когда на определенное время требуется немного свободной памяти, но только во временное пользование.

Однако, системе бейсика принадлежит область памяти до ячейки, указываемой системной переменной `RAMTOP`. По этому адресу находится число *3E (62), которое устанавливает конец используемой бейсиком области. Двигаясь по памяти "вниз", мы сталкиваемся с одним неиспользуемым байтом (этот байт составляет совместно с байтом, указываемым с помощью `RAMTOP`, как бы единое двухбайтовое число (является его младшим байтом), необходимое для верной работы инструкции `RETURN`. Если во время ее выполнения стек `STACK` будет уже пуст, то это число

сыграет его продолжение. Но, поскольку оно больше, чем 15872 (62*256), а строки бейсика не имеют такой высокой нумерации, то это будет восприниматься как ошибка и сигнализировано сообщение "RETURN без gosub". Сразу после этого байта (двигаясь все время "вниз" по памяти) начинается "стек gosub". В него заносятся номера программных строк из которых были выполнены инструкции перехода к подпрограмме, чтобы интерпретатор знал точно, куда должна "вернуться" инструкция return. Если интерпретатор не находится в подпрограмме (вызванной с помощью gosub), то этот стек просто не существует - в нем не записано ни одного значения. Ниже находится "машинный стек", непосредственно используемый микропроцессором, оба этих стека откладываются в сторону уменьшения адресов памяти. Специальную роль играет системная переменная ERRSP. Процедура, обрабатывающая ошибку бейсика (вызывается командой микропроцессора RST 8), помещает значение этой переменной в регистр SP, после чего выполняет RET, считывая таким путем последний записанный на стеке адрес (во время выполнения программы он равен обычно 4867). Под тем адресом в котором находится процедура, выводятся сообщения об ошибке.

Программа на бейсике	(-- PROG (23635) [23755]
Переменные бейсика #80	(-- VARS (23627)
Область редактирования строки бейсика #0D #80	(-- E- LINE (2341)
Буфер INPUT #0D	(-- VDRSP (23649)
Текущая рабочая область Стек I калькулятора v	(-- STKBOT (23651)
Свободно	(-- STKEND (23653)
Машинный стек A I	(-- регистр SP (-- ERRSP (23613)

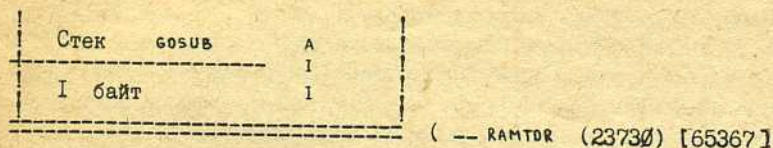


Рис.2. Область памяти системы бейсик

Графика, определенная пользователем

- 168 байт, зарезервированных для определения знаков `ud6` (их можно ликвидировать, например, с помощью `clear 65535`). Адрес последней ячейки памяти (равной 65535, если ваш компьютер полностью исправен) заполняется в переменной `r-ramt`. Если часть памяти `ram` повреждена, то эта переменная содержит адрес последней исправной ячейки.

2. Способ записи программ на ленте.

Помни, что практически каждая программа имеет хотя бы процедуру-загрузчик, написанную на бейсике, а если ее нет, то вообще не защищена.

Начнем со способа записи программ на ленте - припомни, что видно и слышно во время считывания какой-либо программы. В течении 1-2 секунд на экране видны широкие красно-синие полосы, а также слышен длительный звук. Это, так называемый пилот, который позволяет компьютеру синхронизироваться с сигналом с ленты, который он будет считывать. Потом, на момент, появляются тонкие мерцающие желто-фиолетовые полосы, свидетельствующие о том, что компьютер считывает в память информацию. Ее 17 байтов - это, так называемый, заголовок. Появляется надпись "bytes :", "program :", "character array :" или "number array :", а пилот (более короткий), после него считывается собственно программа. Займемся теперь заголовками, так как в них содержатся важнейшие данные о считываемых программах.

Заголовок - как мы уже вспоминаем - содержит 17 байт. Пронумеруем их от 0 до 16 (см.рис.1). Нулевой байт обозначает тип блока. Он равен 0, если это программа на бейсике; 3 - если это блок машинного кода (записанный с помощью "save".... "code или save ".... "screen", которые обозначают точно то же, что и "save".... "code" 16384, 6912). Если же этот заголовок предшествует набору, являющемуся массивом переменных бейсика (записанный с помощью save "...." data), то он равен 1 - для числовых массивов или 2 - для символьных.

Следующие 10 байтов - это имя считываемого блока, или текст, появившийся после загрузки заголовка за надписью "program :", "bytes:" или другой.

В зависимости от типа считываемого блока, байты с I3 по I6 интерпретируются по разному. Начнем от заголовка программ написанных на бейсике. Байты I3 и I4 содержат номер строки старта программы, если она была записана с помощью save "...".LINE nr. Если программа была написана без опции line и после считывания не стартует автоматически, то значение этого числа больше 32767. Одним из способов нейтрализации защиты самостартующих программ на бейсике является замена этих двух байтов в заголовке на число больше 32767.

Байты I5 и I6 содержат число, определяющее длину самой программы на бейсике (так как save "...".LINE записывает программу вместе со всеми переменными, т.е. содержимое памяти от байта, указываемого системной переменной prog, до байта, определенного переменной E-LINE). Если от всей длины блока (байта I1 и I2) отнимем это число, то узнаем, сколько байтов в этом блоке занимают переменные бейсика. Это - все, если речь идет о заголовках программ на бейсике.

В заголовках машинного кода ("bytes") байты I5 и I6 не используются, зато I3 и I4 составляют двухбайтовое число, определяющее по какому адресу надо считать следующий за заголовком блок.

В заголовках массивов из этих четырех байтов используется только I4-й байт, который представляет имя считываемого массива. Он записан так же, как и имена всех переменных бейсика (в области от vars до E-LINE), т.е. три самых старших бита обозначает тип переменной (здесь это числовой или знаковый массив), а 5 младших битов - имя этой переменной.

Ø ! Тип блока:		Ø - программа	
		I	- NUMBER ARRAY (числовой массив)
I !		!	2 - CHARACTER ARRAY (символьный массив)
2 !		!	3 - BYTES (данные)
3 !			
4 !			
5 !	Имя		
6 !			
7 !			
8 !			
9 !			
10 !			
II ! Длина		Сколько байт необходимо будет считать с ленты	
I2 !			

I3	!	Старт	!	BYTES:	адрес загрузки блока
I4	!		!	PROGRAM:	номер строки старта программы
	!		!	ARRAY:	байт I4 - имя переменной
	!		!		I3 - не имеет значения
I5	!	Программа	!	PROGRAM:	длина самого бейсика
I6	!		!		(без переменных)
	!		!	BYTES, ARRAY:	не используется

На листинге I представлена программа, реализующая процедуру, делающую возможным чтение заголовков программ, записанных на ленте. После точного ввода программы (внимательно проверь количество пробелов в строках с инструкциями DATA) запусти ее с помощью `run` и подожди минуту. На экране появится информация об адресах начала и конца процедуры, а также о ее длине, которая должна составлять 284 байта. Если это не так, проверь: все ли строки программы введены без ошибок. Если все данные были верны, то на твоей ленте окажется процедура "чтец", благодаря которой ты сможешь прочесть каждый заголовок. После записи ее на ленту, ее можно удалить из памяти.

Процедура "чтец" будет работать правильно, независимо от того, по какому адресу она будет загружена, ты можешь считать ее с помощью:

`LOAD "czytasz" code адрес`

а затем запустить (даже многократно) с помощью

`RANDOMIZE usr адрес`

После запуска процедура считывает по адресу 23296 (в буфер принтера) первый встречный заголовок. Если из-за подключенных внешних устройств этот адрес не устраивает тебя, то можешь его сменить, заменяя в строке 200 программы с листинга I число # 0058 шестнадцатичным адресом, по которому ты хотел бы считывать заголовки (две первые цифры являются младшим байтом этого адреса). Чтобы после этой замены избежать проверки контрольной суммы в этой строке, в конце текста, взятого в кавычки, вместо пробела и четырех цифр контрольной суммы помести литеру "s" с четырьмя ведущими пробелами.

После считывания заголовка процедура считывает заключенную в нем информацию и возвращается в бейсик, но считанного заголовка не уничтожает, следовательно если желаешь его просмотреть дополнительно, то можешь сделать это, используя функцию `peek`.

Однако, чтения заголовков мало, чтобы суметь взломать блоки, записанные на ленте, необходимо еще знать: что надо сделать с этими блоками,

чтобы разместить их в памяти, не позволяя им при этом начать работу.

В случае блоков типа "bytes" достаточно, обычно, загрузить их под принудительный адрес выше ramtop (т.е. выше ячейки памяти, указываемой системной переменной ramtop), например, с помощью:

```
SLEAR 29999: LOAD " " CODE 30000
```

этот метод работает, если только считываемый блок не очень длинный (может иметь максимально до 40 к). Более длинные блоки могут просто не поместиться в память - тогда необходимо разделить их на несколько более коротких частей. Скоро мы узнаем как это сделать.

Так же и в случае массивов, их загрузка не вызывает затруднений - достаточно применить обычную в таких ситуациях инструкцию `LOAD " " DATA ...`

Хуже, зато, выглядит считывание программ на бейсике. Они обычно записываются с помощью `SAVE "...."LINE`, а в самом начале строки, с которой должны выполняться, размещены инструкции, защищающие программу от останова. Простейшим решением является загрузка программы не с помощью `LOAD " " " " " "`, а с помощью `MERGE " " " " " "`, но этот способ не всегда дает результат. Из этой безнадёжной ситуации существует два принципиальных выхода: подменить заголовок программы или использовать представленную ниже программу "LOAD/MERGE". Первый способ основан на замене записанного на ленте заголовка на такой же, но не вызывающий самозапуск программы. Можно с этой же целью использовать программу "SOURCE" - считать заголовок программы, нажать "BREAK", а затем инструкцией `LET` изменить ее параметр `STAT` на число большее 32767 (т.е. выполнить, например `LET I=32768`, если корректируемый заголовок был считан как первый набор). Модифицированный таким образом заголовок записываем где-нибудь на ленте. Убираем из памяти программу "SOURCE" и вводим `LOAD " " " " " "`. Считываем сделанный только что заголовок, но не сразу после его окончания останавливаем ленту. Теперь в магнитофон вставляем кассету с программой - так чтобы считать только текст программы без его заголовка.

Другой способ намного проще. Вводим в память (с клавиатуры или ленты) программу "LOAD/MERGE", размещенную на листинге 2. После запуска она начинает ждать первую программу на бейсике, которая находится на ленте. Считывает ее совершенно так же, как инструкцию `LOAD " " " " " "`, но после загрузки не позволяет программе запускаться - вводит только сообщение "OK" с информацией с какой строки считанная программа должна стартовать.

На листинге 3 представлена процедура, которая образует эту программу из пакетов. Ее определяющей инструкцией является `CALL I82I`. Начиная с адреса `I54I`, в `ROM` находится процедура, интерпретирующая инструкции `SAVE`, `LOAD`, `VERIFY`, `MERGE`. `CALL I82I` прыгает прямо в середину этой процедуры. Ее начальную часть (считывающая параметры этих инструкций)

обходим заменяя это строками 40-90 нашей процедуры: сначала с помощью RST 48 резервируем 34 байта в области WORKSP. Первые 17 байт заполняются процедурой из ROM после считывания параметров выполняемой инструкции бейсика, а вторые - это место для считанного заголовка. Оба этих заголовка, затем, сравниваются, перед тем как прочесть блок данных (проверяются соответствия и типа блока). Адрес свободного места переносим из регистра DE в IX в строке 80 фиксируем в системной переменной T-ADDR, что нам нужна LOAD, а не, например, VERIFY, а в первом байте заголовка для сравнения помещаем значение 255, означающее, что должен быть считан первым встречным блок (т.е. программа на бейсике), остальное берет на себя процедура из ROM, действующая идентично LOAD "". Разница в выполнении LOAD и "LOAD/MERGE основана на том, что после полного считывания программы, вместо возврата по RET в интерпретатор бейсика, мы переносим значение переменной NEWPC (номер строки, к которой должен быть переход) в PC (номер последней выполняемой строки), а также не допускаем самозапуска программы - выполняем RST 8 с сообщением "OK". Благодаря этому переносу в выведенном сообщении будет информация о номере строки, с которой считанная программа должна была запуститься.

2.1. Листинги программ

Листинг I

```

10 CLEAR 59999:LET POCZ = 60000
20 LET ADR = POCZ
30 RESTORE:READ A,B,C,D,E,F
40 DATA 10,11,12,13,14,15
50 LET NR=200:RESTORE NR
60 LET S=0:READ A$:IF A$="" THEN GO TO 130
70 FOR N=1 TO LEN A$-5 STEP 2
80 LET W=15*VAL A$(N)+VAL A$(N+1)
90 POKE ADR,W:LET ADR=ADR+1:LET S=S+W
100 NEXT N
110 IF VAL A$(N TO)<S THEN "OSH.W STOK":NR:
STOP
120 LET NR=NR+10:GOTO 60
130 PRINT "VSE DANNYE HOROSHO "" "" NACHALO:";
POSZ"" KONIEC: _";ADR-1""DLINA _ _";ADR-POCZ
140 SAVE "CZYTACZ" CODE POCZ,ADR-POCZ
150 REM
200 DATA "21920009E5DD210058DDES1111 1246"
210 DATA "00AF37CD5605DDE130F23E02CD 1531"

```



```

220 DATA "011611C009DD7E00CD0A0CDDE5 1265"
230 DATA "D113010A00CD3C202A7B5CD1E5 1231"
240 DATA "D5ED537B5C010900CD3C20DD46 1346"
250 DATA "0CDD4E0BCD2B2DCDE32DE1DD7E 1664"
260 DATA "00DD460EDD4E0DC5A72026E601 1292"
270 DATA "1900CD3C20D5DD+610DD4E0FCD 1361"
280 DATA "2B2DCDE32DD1C178E6C0206EC9 1848"
290 DATA "011300CD3C20C1CD282DCDE32D 1280"
300 DATA "18FEFE03203601700009EB0111 836"
310 DATA "0018E60D449C75676F9D9B2073 1281"
320 DATA "616D6F676F2070726F6772616D 1313"
330 DATA "7F200D4175746F737461727420 1161"
340 DATA "2D206C696E696120D2EE180 1239"
350 DATA "0009EB010900CD3C20C13E1FA0 997"
360 DATA "F660D7DD7E003D28033E24D73F 1383"
370 DATA "20D73E29D7E1227B5CFE0DD7C9 1338"
380 DATA "04081C2020201C000010181030 268"
390 DATA "100C0008103840380478000D41 430"
400 DATA "64726573209C61646F77616E69 1357"
410 DATA "61200D5461626C69636120 862"
420 DATA " ."

```

Листинг 2

```

1 REM LOAD/MERGE TS & RD 1987
2 FOR N=60000 TO 60025:READ A:POKE N,A:NEXT N
3 RANDOMIZE USR 60000
4 DATA 1,34,0,247,213,221,225,253,54,58,
1,221,54,1,255,205,29,7,42,66,92,34,69,
92,207,255

```

Листинг 3

```

10 ; LOAD/MERGE
20 ;
30 ORG 60000
40 LD BC,34
50 RST 48
60 PUSH DE
70 POP IX
80 LD (IX+58),1
90 LD (IX+1),225
100 CALL 1821
110 LD HL,(23618)
120 LD (23621),HL

```


I30 RST 8

I40 DEFB 255

3. Защита программ на бейсике.

Прочитанная программа, как правило, не должна выглядеть "нормально". Например, в программе есть строка с номером 0 или строки упорядочены по убыванию номеров; нельзя вызвать EDIT ни для какой строки, видно подозрительную инструкцию RANDOMIZE USA 0 или просто ничего не видно, так как программа не позволяет листать себя, если в программе, куда ты вламываешься, ты увидел что-то необычное, то лучше просматривать ее другим, несколько отличающимся от обычного, способом - не с помощью LIST, а непосредственно используя функцию реек.

Однако, сначала мы должны узнать, каким образом размещен в памяти текст программы на бейсике, программа складывается из последовательных строк и так хранится в памяти.

Отдельная строка программы выглядит таким образом:

MSB	LST	LSB	MSB	
2 байта		2 байта		... #0D
Номер строки		Длина		Текст ENTER
		текста + ENTER		

Рис.1.

Она занимает не менее 5 (а точнее, 6, так как текст пустым быть не может) байтов, два первых обозначают ее номер, но он записан наоборот, от всех обычных двухбайтовых чисел, хранимых в памяти (MSB - старший байт, LST - младший байт).

Следующие 2 байта - это длина строки, т.е. число символов содержащихся в строке вместе с завершающим ее знаком ENTER (#0D) за этими байтами находится текст строки, заканчиваемый ENTER.

Если мы введем, к примеру, такую строку:

10 REM BASIC

и запишем ее, нажимая клавишу ENTER, то она будет записана в память как последовательность байтов:

0	I0	7	0	234	66	65	83	73	67	I3
I0		7		REM	B	A	S	I	C	ENTER
Номер		Длина		Текст						

Рис.2.

Параметр "длина строки" касается только ее текста, следовательно, хотя строка занимает в памяти 11 байт, этот параметр указывает лишь на 7 байт: 6 байт текста и 1 байт - "ENTER", заканчивающий строку.

Тебе, наверное, уже понятно на чем основан часто применяемый трюк со строкой, имеющей нулевой номер, достаточно в первые два байта строки занести число 0 (с помощью роке), чтобы эта строка стала нулевой строкой. Если мы хотим изменить номер первой строки в программе (а интерфейсы никакой быстрой памяти не подключены, т.к. в этом случае изменяется адрес начала бейсика), то достаточно написать:

Роке 23755, X: РОКЕ 23756, Y

и строка получит номер $256 * X + Y$. Независимо от его значения строка остается в памяти там, где была, если, к примеру, введем:

10 REM номер строки 10

20 REM номер строки 20

РОКЕ 23755, 0: РОКЕ 23756, 30

то первой строке в программе будет присвоен номер 30, но она останется в памяти как первая, а на экране мы получим следующее:

30 REM номер строки 10

20 REM номер строки 20

Следовательно, чтобы начать разблокировать программу, в которой имеются нулевые строки, упорядоченные по убыванию номеров, следует найти адреса каждой строки и в их поле "номер строки" последовательно размещать, к примеру, 10, 20, 30, ... В памяти строки располагаются одна за другой, следовательно, с обнаружением их начал ты будешь иметь трудности. Если X указывает адрес какой-нибудь программной строки, то следующий адрес равен:

$X + \text{PEEK}(X+2) + 256 * \text{PEEK}(X+3) + 4$

т.е. к адресу строки добавляется длина ее текста, увеличенная на 4 байта, т.к. именно столько занимают параметры "номер строки" и "длина строки".

Такой способ нахождения начала строки не действует к сожалению, когда применяется другой способ защиты - фальшивая длина строки. Он основан на том, что в поле "длина строки" вместо настоящего значения дает очень большое число - порядка 43-65 тысяч. Этот способ применяется очень часто, так как обычно возможно считывание программы с помощью моке (т.е. так, чтобы не было самостарта). Делается это потому, что моке загружает программу с ленты в область WORKSPACE, а затем интерпретатор анализирует всю считанную программу строка за строкой: последовательно номер каждой из них, а затем размещает в соответствующем месте области, предназначенной для текста программы на бейсике. Для этой строки необходимо там подготовить соответствующее количество свободных байт, "раздвигая" уже существующий текст программы. Если в поле "длина строки" стоит очень большое число, то интерпретатор будет стараться сделать именно столько байтов пространства в области текстов программы, что это завершится сообщением "out of

MEMORY " или просто записываем системы. Чтобы прочесть такую программу, не вызывая ее самозапуск, следует применить соответствующую отмычку, например, такую как представленная в разделе 2 программа "LOAD/MERGE".

Дополнительным эффектом от применения фальшивой длины является невозможность корректировки такой строки путем занесения ее в поле редактора с помощью клавиши "EDIT". Ситуация выглядит аналогично: операционная система старается освободить место для этих строк в области редактирования строк бейсика (от переменной `E..LINE DO WORKSP`). Однако, это требует слишком большого количества свободной памяти, следовательно, кончается только предупредительным сигналом.

Если программа защищена этим способом, то адреса очередных строк приходится искать "вручную" или догадываться, где они находятся, помня о том, что каждая строка кончается знаком "ENTER" (но не каждое число 13 означает "ENTER").

Чтобы просмотреть программу на бейсике, введи такую строку:

```
FOR N=23755 TO PEEK 23627+256*PEEK 23628:  
PRINT N; ";PEEK N,CHR& PEEK N AND PEEK N > 31:  
NEXT N
```

она последовательно высветит: адрес, содержимое байта с этим адресом, а также символ, имеющий этот код, если только это не управляющий символ (т.е. с кодом 0...31).

После смены нумерации строк и изменения их длин, следующим способом защиты программ являются управляющие символы, не позволяющие последовательно просматривать программу, хотя и не только это.

Вернемся к первому примеру (строка "10 REM BASIC").

Текст строки складывается из 7 символов - ключевого слова `REM` (все ключевые слова - инструкции, функции, а также знаки `=`, `,` и `()` имеют однобайтовые коды из диапазона 165...255. Если ты не знаешь какие, то введи:

```
FOR N=165 TO 255:PRINT N,CHR& N:NEXT N
```

и ознакомься), а также пяти литер и символа "ENTER". Так бывает всегда, если в строке находится инструкция `REM` - все знаки, введенные с клавиатуры после этой инструкции, будут размещены в тексте строки без малейших изменений. Иначе выглядит ситуация, когда в строке находятся другие инструкции, требующие числовых параметров (а обычно так и бывает). Введем, например, строку.

```
10 PLOT 10,9
```

и посмотрим, каким образом она запишется в память (лучше - вводя приведенную выше строку `FOR N=23755 TO...`). Выглядит она так, как показано на рис.3:

(< обозначает CHR& 8), то инструкция LOAD " " и последующий текст закроют предшествующую инструкцию LET A=USR 0:. Хотя на листинге теперь видна только инструкция LOAD " ", но дальнейшая часть программы не загружается ею, а машинной программой, запускаемой функцией USR 0, (что, очевидно, не должно обозначать переход к адресу 0). Такая защита, например, уточняющий о каком атрибуте идет речь. После символов INK и PAPER это могут быть числа 0 ...9, после FLASH и BRIGHT : 0, 1 и 8, после INVERSE и OVER : 0 и 1. Задание других значений вызывает сообщение : "INVALID colour " и , естественно, прерывание просмотра программы.

Высвечивая программу, защищенную управляющими символами цветов, принимает на себя следующую последовательность действий, например, если, просматривая текст программы мы встречаем код знака PAPER CTRL , то заносим в его второй байт значение 0, если INK CTRL - значение 7, в остальные управляющие цветами символы - значение 0. Кроме того, удаляем все мешающие знаки BASKR&8 (CHR& 8) путем их замены пробелами (CHR& 32). Также ликвидируем знаки AT CTRL - заменяем с помощью REKE три байта символа на пробелы . После такой корректуры программу уже можно листать без всяких неожиданностей.

Если ты хочешь взломать программу-загрузчик, то его необязательно, а в принципе и не надо очищать - важно узнать, что эта программа делает, каким образом загружает в память и запускает следующие блоки, а не стараться, чтобы это она делала "ладно" и была написана чисто и прозрачно. Это тем более важно, что пока не узнаешь точно программу, лучше не делать в ней никаких изменений - одна ловушка может проверяться другой. Поэтому наилучшим способом "раскалывания" программы является анализ ее работы за шагом, считывая последовательные байты памяти:

```
0 BORDER 0: PAPER 0: INK 0: CLS: PRINT "0, "LOADING":
FOR N=0 TO 20 STEP 4: BEEP .2,N: NEXT N: LOAD""CODE:
PRINT AT 19,0;: LOAD""CODE: PRINT AT 19,0::
LOAD""CODE: PRINT AT 19,0;: LOAD""CODE: PRINT AT 19,0;
: LOAD""CODE: RANDOMIZE USR 24064
```

Помни о правильной интерпретации очередных байтов: сначала 2 байта номера строки, потом 2 байта содержания ее длины, (которая может быть фальшивой), затем текст: инструкция бейсика, потом ее параметры, за каждым числом дописываются CHR& 14 и пять байтов, содержащих значение этого числа. За параметрами - двоеточие и следующая инструкция или "inter" и новая строка программы.

Это было бы все, если речь идет об управляющих символах. Но есть еще одна вещь, которую требуется объяснить, чтобы не иметь неприятностей с чтением бейсика. Речь идет об инструкции DEF FN. ВВЕДИ , а затем

внимательно посмотри такую строку:

```
10 DEF FN A(A<B&<C)=A+C
```

Кажется, что она должна занять в памяти 19 байтов (номер строки, ее длина, ENTER, а также введенных символов), но это не так. Интерпретатор после каждого параметра функции поместил знак CHR& I4 и дорезервировал зачем-то следом еще 5 байтов. Введи:

```
PRINT FN A (1, "125", 2)
```

И снова внимательно просмотри содержимое памяти с адреса 23755. После первого параметра в определении функции далее находится CHR& I4, но после него последовательно расположились: 0, 0, 1, 0, 0, что в пятибайтовой записи означает число 1. Также, после третьего параметра функции находится CHR& I4 и байты, содержащие число 2. После параметра & также находятся значение использованного параметра: CHR& I4 и пять байтов, которые последовательно содержат: первый для нас не имеет значения, второй и третий содержат адрес, по которому находилась цепочка символов "123" (вызов функции был осуществлен в директивном режиме, следовательно, этот адрес относится к области редактирования строки бейсика), а байты 4 и 5 - это длина цепочки - в нашем случае она составляет 3 знака.

Помни об этом, читая бейсик с помощью реек, а не list. Иногда случается, что именно в этих байтах, зарезервированных для действительных аргументов функции скрыты проверки, определяющие работоспособность программы или даже машинная программа, загружающая последующие блоки, например, BETA BASIC I.0).

В конце немного о программах-загрузчиках. Их задачей является считывание и запись всех блоков, составляющих программу. Обычно они это делают способом, максимально затрудняющим понимание их работы - так, чтобы запуск программы другим способом, а не через его загрузчик (или на практике взлом программ) был невозможен. Посмотрим на загрузчики, применяемые в большинстве продукции фирмы ULTIMATE (например, ATLAS, KNIGHTLORE, NIGHT SHADE и т.д.). Выглядят они примерно так:

```
FOR N = 23755 TO REEK 23627 + 256*реек 23628 :
```

4. "Защитные" управляющие символы.

В этом разделе мы познакомимся с остальными управляющими символами. Три из них касаются места вывода, шесть - смены атрибутов.

4.1. CHR& 6 - "COMMA CONTROL"

Этот символ (управляющий запятой) действует также как запятая, отделяющая тексты в инструкции PRINT, т.е. выводит столько пробелов (но всегда не менее одного), чтобы оказаться или в колонке 0 или 16:

PRINT "I", "2"

а также

PRINT "I" + CHR& 6+ "2"

имеют идентичное значение.

4.2. CHR& 22 - "AT STRL "

Этот символ (AT управляющий) позволяет переносить позиции вывода в любом месте экрана также, как AT в инструкции PRINT. После этого знака должны появиться два байта, определяющие номер строки и номер колонки, в которой должен быть расположен следующий знак:

PRINT AT 10, 7; "!"

равнозначно

PRINT CHR& 22; CHR& 10; CHR& 7; "!"

Чтобы убедиться как с помощью этого символа сделать листинг программы невидимым, введи например:

10 RANDOMIZE USR 30000:REM ничего не видно!

После инструкции REM введи три пробела, а после восклицательного знака две управляющих запятых. Их можно получить непосредственно с клавиатуры, нажимая последовательно клавиши: EXTEND (или оба SHIFTa вместе), чтобы получить курсор "E", а затем клавишу "6" (курсор сменит цвет на желтый и DELETE - курсор перескочит к ближайшей половине экрана.

После ввода этой строки, заменим три этих пробела на знак AT 0, 0 с помощью:

POKE 23774, 22; POKE 23775, 0; POKE 23776, 0

Попробуем теперь посмотреть программу. На экране не появится текст всей строки - начальная ее часть закрыта надписью, находящейся после инструкции REM и знака AT STRL. Такие же трудности возникнут, если эту строку перенести в зону редактирования (клавиша EDIT).

Координаты, заданные в символе AT STRL должны находиться в поле экрана, т.е. номер строки не может быть больше 21, номер колонки - больше 31. Задание больших значений в случае PRINT и LIST вызывает сообщение "out of screen" и отмену дальнейшего вывода. Если листинг получен нажатием "ENTER" (автоматический листинг) - также наступит прекращение дальнейшего вывода, а кроме того в нижней части экрана появится мигающий знак вопроса - сигнал ошибки. Следовательно, это практически способ защиты от просмотра текста для каждой программы.

4.3. CHR& 23 - "TAB STRL "

После этого символа (горизонтальная табуляция) следуют два байта, определяющие номер колонки, в которую переносится позиция вывода. Они трактуются как одно двухбайтовое число (первый байт - младший). Поскольку

ку колонок только 32, . То число берется по модулю 32, т.е. старший байт и три старших бита младшего байта игнорируются. Второй существенной стороной является то, что TAB переносит позицию вывода с помощью вывода пробелов - так же и управляющая запятая, и, следовательно, может быть использован для закрывания уже находящихся на экране текстов.

4.4. Символы смены атрибутов

Эту группу управляющих символов составляют символы, имеющие атрибуты:

```
CHR& I6 - INK CTRL  
CHR& I7 - PAPER CTRL  
CHR& I8 - FLASH CTRL  
CHR& I9 - BRIGHT CTRL  
CHR& 20 - INVERSE CTRL  
CHR& 21 - OVER CTRL
```

После каждого из этих символов обязателен один байт,
PRINT N;" "; PEEK N, CHR& PEEK N AND PEEK N > 31:
NEXT N

После такого загрузчика на ленте находится 5 следующих блоков: экран, закодированный главный блок программы, а за ними 3 коротеньких блока, защищающих программу: однобайтовый (код инструкции JP (HL)), из нескольких байтов (это процедура, которая декодирует всю программу) и последний двухбайтовый, загружаемый по адресу 23672, или в переменную FRAMES. Значение этой переменной увеличивается на 1 через каждые 1/50 секунды. В машинной программе, запущенной с помощью RANDOMISE usr 24064, ее значение проверяется, и, если отличается от того, каким должно быть (что означает, что где-то после загрузки программа была остановлена на какое-то время), наступает обнуление памяти компьютера. Взлом программ этого типа весьма прост. Достаточно загрузить все блоки, за исключением последнего, а после просмотра программы, или же проведения в ней определенных изменений (например, вписание реке), достаточно лишь ввести:

```
LOAD" "CODE:RANDOMISE usr 24064
```

(Обязательно в одной строке, разделяя инструкции двоеточием), чтобы запустить игру. Lin 0 отдельной работой является декодирование программ, или запуск процедуры, декодирующей так, чтобы по завершению работы вернулась в бейсик. У читателей, знающих ассемблер, это не должно вызывать затруднений, однако с точки зрения на распространенность этого типа защиты, особенно в пользовательских программах (например, art studio или the last word), мы еще вернемся к этой.

5. Защита загрузчиков

Все игры имеют очень хорошо защищенную программу, написанную бейсиком, так как это важнейший (с точки зрения действительности защиты) элемент всей программы - ведь с бейсика начинается считывание всей программы. Если бейсиковый загрузчик защищен слабо, то взлом всей программы значительно облегчен, примером этого являются загрузчики фирмы ultamate, представленные в предыдущем разделе. Одним из способов снятия защиты загрузчиков является считывание их с помощью программы "load /merge" (см.раздел 2). Однако иногда лучше поместить этот загрузчик не в памяти, предназначенной для бейсика, а выше ramtor, чтобы можно было спокойно просматривать его, не опасаясь возможности случайных изменений в нем.

Для этого имеется очень действительный метод - считывание программы на бейсике как блока машинного кода под удобный для нас адрес. Чтобы этого добиться требуется знать длину программы, которую мы хотим считать (можешь использовать процедуру "szytasz" из раздела 2) хотя можно обойтись и без длины, кроме того, требуется немного свободного места на магнитной ленте. Этот способ основывается на обмане инструкции load путем подмены заголовков.

На свободной ленте записываем заголовок блока кода с помощью save "bas" code 30000,750, если знаем, что длина программы составляет 750 байтов. Если ее не знаем - подаем соответственно большее значение даже порядка нескольких килобайт, хотя программа может иметь всего лишь 100 байтов длины. На ленте записываем только сам заголовок, прерывая запись после этого нажатием клавиши "break". Теперь устанавливаем ленту точно перед записанным заголовком, а ленту с программой - сразу после заголовка программы, но перед требуемым блоком данных. Вводим:

```
CLEAR 29999: LOAD " " CODE
```

или

```
CLEAR 29999: LOAD " " CODE 30000
```

и считываем заголовок. Сразу после его считывания, мы нажимаем stop в магнитофоне, заменяем кассету и вновь нажимаем runk (все это время компьютер ждал блока данных). Теперь считывается программа на бейсике. Но под адрес 30000 - выше ramtor. Если мы задали в заголовке завышенную длину программы, то считывание заканчивается сообщением "TARE LOADING ERROR", но это не мешает - теперь уже любым способом можно смотреть считанную программу, даже набирая с этой целью программу на бейсике.

Кроме этого метода существует и второй, но для того, чтобы им воспользоваться, обязательно знание ассемблера (а пользоваться стоит, так как дает он больше возможности в раскрытии программ, а значение ее позволяет обычно расшифровывать работу загрузчика).

Очень часто (особенно в новейших программах) встречаются блоки программ, написанные и считываемые в память компьютера без заголовка. Это достаточно оригинальная и эффективная мера защиты, обычно отпугивает начинающих, но раскрытие такой программы вовсе не является трудным. Вся тайна основана на использовании хранящихся в ROM SPECTRUM процедур, используемых с помощью инструкций LOAD, SAVE, VERIFY и MERGE.

Под адресом ± 0556 (I366) хранится процедура LOAD-BYTES, считывающая с магнитофона блок данных, или пилота и следующую за ним информацию. При этом не важно будет ли это заголовок, или требуемый блок данных, которые следует разместить где-то в памяти.

Начнем все же сначала. Каждая защищенная программа начинается с загрузчика, написанного на бейсике. Программа, применяющая загрузку без заголовков (с помощью процедуры I366 или другой) должна быть написана в машинном коде, как каждая процедура, обслуживающая магнитофон. Чаще всего эта программа помещается в одной из строк бейсика, например, после считывания, загрузчик на бейсике запускается и выполняет инструкцию RANDOMISE usr иницируя тем самым работу машинной программы.

Процедура load-bytes требует соответствующих входных параметров. Они передаются в соответствующих регистрах микропроцессора. Так, в регистре IX задаем адрес, под которым хотим прочесть блок данных, а в паре DE - длину этого блока. В буфер помещаем 0, если хотим считать заголовок и 255, если этот блок данных. Кроме того, указатель переноса (CARRY) устанавливаем в 1, так как процедура I366 вместо load выполнила бы функцию verify. Ниже дан пример процедуры, загружающей с ленты образ без заголовка:

LD IX,16384;	адрес считывания
LD DE,6912;	длина блока
LD A,155;	блок данных
SJF;	установка CARRY
CALL I366;	вызов LOAD-BYTES
RET	выход из подпрограммы

Процедура I366 в случае ошибки считывания не выводит сообщения "TARE LOADING ERROR". Но существует еще одна процедура загрузки, которая это делает. Она находится под адресом 2050 и выглядит так:

2050 CALL I366: считывание блока данных


```
2043 RET C; возврат если не было ошибки
2054 RST 8; "TARE LOADING ERROR"
2055 DEFB 26; "TAPE LOADING ERROR"
```

После возврата из процедуры I366 указатель переноса содержит информацию о правильности считывания блока,. Если он удален, то это означает, что наступила ошибка. Некоторые загрузчики используют именно процедуру 2050, а не I366.

Иногда загрузчики не пользуются ни той, ни другой процедурами, а заменяют их соответственной, но она, однако, обычно очень похожа на процедуру I366 или даже является ее переделкой, благодаря которой, блоки данных загружаются в нижнюю часть памяти - с больших адресов к нижним, или, например, загрузка идет с другой скоростью. Такую программу следует анализировать с помощью дисассемблера (например, *mons*), сравнивая некоторые ее фрагменты с тем, что находится в *rom*.

Сейчас мы объясним, как использовать процедуры из *rom* для считывания бейсика под любой адрес, а не в область, предназначенную для него с помощью "считасз" прочитаем заголовок программы, которую мы хотим вскрыть, и запоминаем ее длину(т.е. длину всего блока - программу вместе с переменными). Затем вводим соответствующую программу на ассемблере, которая прочтет бейсик по адресу, который мы установим (выше *ramtor*):

```
LD IX, адрес
LD DE, длина
LD A, 255
SCF
JP 2050
```

Так же как и при подмене заголовка, если мы не знаем длины программы, то можем задать завышенное значение, то тогда чтение завершится сообщением "TARE LOADING ERROR". Но считывание ассемблера каждый раз, чтобы ввести программу, приведенную выше может вызвать раздражение, следовательно, лучше создавать эту программу с уровня бейсика с помощью роке:

```
10 INPUT "адрес чтения basic ?";A
20 RANDOMIZE A: CLEAR A-I
30 LET A=peek 23670: LET v=peek 23671
40 LET AND=256*B+A
50 INPUT "длина basic ?":C
60 RANDOMIZE M: LET C=peek 23670
70 LET D=peek 23671
80 FOR N=AND TO ADR+44
90 READ X: POKE N,X
```


100 NEXT N

110 DATA 221, 33, A, B, 17, C, D, 62, 255, 195, 2, B

120 RANDOMIZE USR ADR

Устанавливаем ленту с обрабатываемой программой за ее заголовком. Затем запускаем программу, приведенную выше, вводим данные и включаем магнитофон. Результат аналогичен тому, который получаем при подмене заголовков, но первым же видимым достоинством этого способа является то, что мы не создаем беспорядок на кассетах.

В завершение стоит вспомнить еще об одной процедуре, размещенной в ром под адресом 1218. Это процедура `save-bytes`, обратная `load-bytes`, т.е. записывающая на ленту блок с заданными параметрами: перед ее вызовом в регистре IX размещаем адрес, с которого начинается запись, DE содержит длину записываемого блока. В буфере помещаем должен ли это быть заголовок (0) или блок программы (255). Состояние ука

6: Использование системных процедур

В предыдущем разделе были представлены процедуры из ром: `save-bytes` и `load-bytes`. Здесь рассказывается как эти процедуры используются для защиты программ. Займемся блоками машинного кода, которые запускаются удивительным образом.

Первым таким способом является прикрытие загрузчика блоком, который им загружается. 0 такая защита, к примеру, применяется в игре "Три недели в Парадиз-сити". Проследим способ чтения этой программы, так чтобы она не стартовала автоматически.

Начнем, как обычно, с бейсика. Оказывается, практически, единственно важной инструкцией является `randomize usr`. Лучше всего восстановить эту процедуру начиная с адреса запуска `randomize usr` (реек 23627+256*реек 23628), т.е. в нашем случае с адреса 24130. Подобные процедуры используют известную нам процедуру 1366, считывающую блоки без заголовка. Так и в этом случае, но перед ее вызовом с помощью команды `ldix`, процедура загрузки переносит себя в конец памяти (под адрес 63116) и переходит туда по команде `jr`:

24130 DI ;	запрет прерываний
24131 LD SP, ;	LD SP 65536
24134 LD HL, (23627);	В HL адрес на 28
24137 LD DE, 28 ;	больше чем значение
24140 ADD HL< DE ;	переменной vars
24141 LD DE, 63116 ;	В DE адрес A B
24144 LD BC, 196 ;	BC длина блока

24I47	LDIR	;	X	X
24I49	JP	63II6	;	продолжение выполнения XXX

24I52	LD	IX, I6384	;	адреса
24I56	LD	DE, 69I2	;	

...

теперь считывается картинка, а затем главный блок данных:

63II6	LD	IX, I6384;	подготовка загрузки
63I20	LD	DE, 69I2 ;	картинки на экран
63I23	LD	A, 255 ;	с помощью процедуры
63I25	SCF ;		LOAD-BYTES из ROM
63I29	LD	IX, 26490 ;	параметры главного
63I33	LD	DE, 38582 ;	блока программы,
63I36	LD	A, 255 ;	который считываясь
63I38	SCF ;		затирает эту процедуру
63I39	CALL	I366 ;	считывание блока

63I42	JP	NH, +79 ;	после возврата из
63I44	CP	A;	процедуры I366 здесь
63I46	CALL	65I9I	уже находится другая
63I49	JP	NC, -2;	программа

Но способ запуска считанной программы требует пояснения.

Как вы знаете, каждая инструкция CALL заносит в машинный стек адрес, с которого начинает работать программа после выхода из под-программы. В этом загрузчике после выполнения второй инструкции CALL I366 в стек заносится адрес команды, следующей за CALL т.е. 63I42 и процедура загрузки обычным способом затирает сама себя, т.к. считывает байты с магнитофона в ту область памяти, где она была размещена. Существенное значение имеет способ запуска считываемой программы: процедура I366 кончается, естественно, инструкцией RET, которая означает переход по адресу, записанному в стек, или, в нашем случае, по адресу 63I42. В процессе считывания программы, процедура, которая находилась там, была заменена считанной программой, но микропроцессор этого не замечает - он возвращается по адресу с которого выполнен CALL I366, не обращая внимания, что там находится уже совершенно другая программа. Это схематично представлено на рис. I.

программа-загрузчик

загружаемая программа

* LD	A, 255		63I36		(254)	
*			63I37		(254)	
* SCF			63I38		(75)	

*! CALL I366	!	63I39	!	(254)	!
*!	!	63I40	!	(255)	!
*!	!	63I4I	!	(255)	!
! JP NZ ,+79	! !	63I42	/->!	DI	!
!	!	. . .	!	LD SP,0	!*
! CP A	!	!-----!	!		!*
! CALL 65I9I	!	!LOAD-BYTES!	!		!*
!	!	! I366 !	!	LD HL,28267	!*
!	!	!-----!	!		!*
-----			!		!* -----

Рис.1.

С левой стороны расписано содержимое памяти до, а с правой - после считывания программы. Команды, отмеченные звездочками (*), ложатся на выполняемую программу.

Возникает вопрос как распознать защиту данного типа и как ее ликвидировать. Начинаем, естественно, с бейсика, считываем загрузчик (в ассемблере) и определяем адреса окончаний считанных блоков (добавляя к адресу начала регистр IX длина блока / регистр DE). Если какой-либо из блоков накрывает процедуру загрузки, то это означает, что программа считывается и запускается именно этим способом.

Дальше все просто. Достаточно, опираясь на данные о блоках (адрес и длину), написать коротенькую процедуру, загружающий нас когда или подготовить соответствующий заголовок, затем с помощью CLEAR ADDR установить соответствующим образом машинный стек (чтобы считываемая программа не уничтожила стек) и, наконец, считать программу. После выполнения в ней необходимых изменений записываем ее на ленту, но таким же образом, каким был записан оригинал (длина должна совпадать прежде всего). Если этот блок был без заголовка (а так и есть в нашем случае), то записываем его обычным SAVE "..." CODE ... , но опускаем заголовок, т.е. включаем магнитофон только в перерыве между заголовком и блоком кода. Также можно пробовать запускать считанный блок переходом на требуемый адрес командой RANDOMIZE USER ... , но это не всегда может получиться. В игре "Три недели в Парадиз-сити" этим адресом 63I42 и как ты можешь убедиться - этот метод срабатывает.

Другим интересным способом запуска блоков машинного кода является считывание программы в область машинного стека. Этим способом можно запускать блоки машинного кода, загружая их просто через LOAD " " CODE. Этот метод схематично представлен на рис.2:

машинный стек				
65536	!			!
65537	!	.	.	!
REG.SP ---->	!	I343		!
65359	!			!
65360	!	2053		!
65361		--		
65362	!	7030	!	65368 !
65363	!	!		!
65364	!	4867	!	? !
65365	!	!		!
65366	!	?	!	? !
RAMTOP ---->	!	62	!	62 !
65368	!	!	LD IX,	!
65369	!	!	...	!
65370	!	!	LD DE	!
		!	...	!

Рис.2.

Указатель стека (регистр `sp`) принимает показанное на рис.2 состояние процесса выполнения процедуры I366 (вызванной из бейсика через `load "code"`). Способ запуска программы суммарно прост. Адрес считывания блока рассчитан так, что блок считывается на машинный стек, именно с того места, в котором находится (записанный интерпретатором бейсика) адрес возврата из инструкции `load "code"` (он тогда равен значению системной переменной `error -2`) или прямо из процедуры `load-bytes`

(равный `error 06`). Тогда 2 первых байта программы обозначают адрес ее запуска. Этот способ очень похож на предыдущий, за исключением того, что там подменялась процедура загрузки, а здесь - адрес возврата из этой процедуры или просто адрес возврата из инструкции `load` . После считывания блока кода микропроцессора считывает содержимое стека и проходит по прочитанному адресу (который только что появился в памяти вместе со считанной). По этому адресу в программе находится начало процедуры загрузки ее последующих блоков - это видно на рис.2.

Метод обхода этой защиты также весьма прост. Достаточно заменить `ramtop` на соответствующее низкое значение, а затем прочитать блок кода, который благодаря этому не запустится. Ситуация может осложниться если блок очень длинный (что случается редко, но встречается) - тогда мы

должны поступить с ним так же, как и каждым длинным блоком, но помнить с какого адреса он запускается.

Займемся теперь расчленением блоков кода с длиной, превышающей 42к. Взлом блоков этого типа основан на разделении их на такие фрагменты, чтобы в памяти еще осталось место для MONS-A или другого реассемблера, исправление этих фрагментов, а затем их "склеивание" в одно целое либо написание новой процедуры загрузки. Обычно достаточно разделить длинный блок на две части. Чтобы получить первую, используем процедуру I366, но с другими параметрами (не с теми, которых требует разделенный на части блок. Ранее из процедуры загрузки или если такой нет, то из заголовка этого блока получаем его длину и адрес загрузки). Просто задаем адрес, по которому хотим разместить этот блок (выше ramtop), а также длину примерно 16 к (несмотря на то, что блок этот значительно длиннее). Считываем теперь этот блок через call I366 или call 2050, но во втором случае сообщение "tape loading error", которое появится не даст нам никакой информации о верности считывания - загружается часть блока и, следовательно, без контрольного байта, который находится в конце. Считанную таким образом первую часть блока записываем на ленту и сразу же приступаем ко второй части. Ее считывание труднее, но тоже возможно, несмотря на ограничения по памяти. Достаточно использовать тот факт, что в spectrum существует 16 к ram, запись в которое просто невозможна. Например, вызываем процедуру I366 с адресом чтения 0 и начальное 16 к считываемого блока будет потеряно, а в память ram считаются только следующие 32к или меньше (в зависимости от длины блока). Считываемый блок займет в памяти ram адреса с I6384 и дальше, заходя на системные переменные и оставляя без изменения только те байты, адреса которых больше длины блока. Поэтому необходимо позаботиться о том, чтобы машинный стек, а также написанную нами процедуру загрузки разместить в конце памяти. Надо также помнить о том, что система бейсика будет уничтожена, и записать сразу считанный блок на ленту можно только процедурой, написанной на ассемблере. Кроме того, в промежутках времени между считыванием фрагмента блока и его записью нельзя разблокировать прерывания, так как они изменяют содержимое ячеек с адресами 23552-23560, а также 23672-23673, а там находится считанный блок. Чтобы выполнить это последнее условие, войдем в середину процедуры I366, благодаря чему после считывания блока не будет выполнена процедура I343. Именно она еще и разблокирует прерывания. С помощью clear 64999 переносим машинный стек, а с адреса 65500 помещаем процедуру загрузки:

ORG 65000 ;

	LD	IX,0;	адрес считывания
	LD	DE,DL;	длина блока
	LD	A,255;	подготовка к
	SCF		считыванию блока
	INC	D;	таким способом
	EX	AF,AF ;	заменяем начало
	DEC	D;	процедуры I366,
	DI;		а затем входим
	LD	A,I5;	в ее середину:
	OUT	(254),A;	
	CALL	I378;	
	LD	A,0;	черная рамка
	JP	C,OK;	будет означать
	LD	A,7;	верное считывание
OK	OUT	(254),A;	белая -ошибочное
CZEKAJ	LD	A,I9I;	ожидаем нажатия
	IN	A,(254);	"ENTER"
	RRA;		
	JR	C,CZEKAJ	
	LD	IX,0;	запись считанного
	LD	DE,DL -I6384;	блока кода на
	LD	A,255;	ленту
	CALL	I2I8;	а также
	HL	,64999	инициализация
	JP	4633;	система бейсикаX

Вместо того, чтобы считывать ассемблер и вводить эту программу, можно запустить программу на бейсике, представленную на листинге I:

Листинг I

```

10 CLEAR 64999
20 INPUT "ДЛИНА БЛОКА?";DL
:RANDOMIZE DL: LET X=PEEK 23670
: LET Y=PEEK 23671: LET S=0
30 FOR N= 65000 TO 65053: READ A
: LET S=S+A: POKE N,A: NEXT N
40 IF S<> 5254 + 2*(X+Y)-64 THEN
PRINT "ОШИБКА В ДАННЫХ": STOP
50 PRINT "ВСЕ ДАННЫЕ ОК" " WKLUCHI MAGNITOFON"
60 RANDOMIZE USR 65550
70 DATA 22I,33,0,0,I7,X,Y,62,2
55,55,20,8,2I,243,62,I5,2II,254

```


80 Дата 205,98,5,62,0,56,2,62,7,
211,254,62,191,219,254,31,56
90 Дата 249,221,33,0,0,17,X,Y-64,62
255,205,194,4,33,231,253,202,25,18,

Как выглядит разделение блока? Устанавливаем ленту на блоке кода, который желаем разделить. Если он имел заголовок, то его опускаем. Запускаем процедуру и включаем магнитофон. Не пугайся, если в определенный момент ты увидишь, что программа считывает на экран - так должно быть. После загрузки блока цвет рамки сигнализирует правильность считывания: если рамка черная, то все в порядке, если белая - была ошибка. Тогда вложи в магнитофон другую кассету, включи запись и нажми "ENTER". Программа запишется на ленту, а потом процедура вернется в бейсик, инициализируя систему сообщением "(C)1982...", но не очищая память (уничтожается только область от начала экрана до примерно 24000 адреса). Теперь, подготавливая заголовок или записывая коротенькую процедуру, можно прочесть полученный блок под любой адрес.

Когда ты найдешь то, что искал и захочешь запустить измененную программу, то придется немного помучиться и "склеить" разделенную программу или написать для нее новую процедуру загрузки. Если программа заполняла полностью 48к памяти ram, возможен лишь второй метод.

Если надо соединить блоки - достаточно написать процедуру похожую на разделяющую, но такую, которая считывает первый блок под адрес I6384, второй - сразу за ним, а затем запишет их вместе как один блок.

В этом разделе был описан способ запуска блоков машинного кода путем считывания в область машинного стека. Чтобы такой блок запустился достаточно ввести инструкцию load "code", которая загрузит его. Чтобы убедиться в этом на практике введи и запусти программу с листинга 2:

Листинг 2

```
10 CLEAR 65361: LET S=0
20 FOR N=65362 TO 65395: READ A
   : POKE N, A: LET S=S+A: NEXT N
30 IF S<>3437 THEN PRINT "OSHIBKA V STROKE DATA"
   : STOP
40 SAVE "BEER" CODE 65362,34
50 Дата 88,255,3,19,0,62,221,2,
   29,6,8,197,17,30,0,33,112,5,205
60 Дата 181,3,33,0,8,43,124,18
   1,32,251,193,16,235,221,225,201
```


Она запишет на ленте короткий блок машинного кода, который будет запускаться самостоятельно. После записи этого блока освободи память с помощью `RANDOMIZE USR 0` или `RESET` (по возможности переставь `RAMTOR` на нормальное значение с помощью `clear 25367`) и прочти его вводя `load code`. Цель этого блока проинформировать о том, что он запустился - он это делает несколькими звуковыми сигналами. Ты услышишь их сразу после считывания программы как только с рамки исчезнут

7. Декодирование закодированных блоков типа "bytes"

Что означает то, что программа или блок закодированы?

Кодирование это род весьма простого шифра, делающего невозможной правильную работу программы. Для ее запуска служит специальная декодирующая процедура, которая находится в этой же программе, а также, что самое важное, не закодирована. Так для чего же служит кодирование? Это просто очередное затруднение доступа к тексту программы после того, как все предшествующие предохранители взломаны и программа считана без запуска. В этом случае в памяти лежит "полуфабрикат", который только после переработки декодирующей процедурой становится программой. Кодирование может быть основано, например, на инверсии всех байтов в программе. Хорошим примером является программа "арт studio". Ее байсовая часть практически не защищена никак, но главный блок программы ("studio - mc" code 26000, 30672) частично закодирован. Что сделать, чтобы раскодировать его? Просто найти адрес, с которого этот блок запускается. В случае "арт studio". То адрес - 26000. Там находится инструкция `jr 26024`, которая осуществляет переход к декодирующей процедуре. Вот ее текст:

26024	2ICI65	LD HL ,26049;	запись адреса
26027	E5	PUSH HL;	26049 в стек
26028	2ICI65	LD HL ,26049	адрес начала
26031	II476C	LD DE ,27719	адрес конца
26034	7E	LD A,(HL);	выбор байта из
26035	D622	SUB 34;	памяти, переко-
26037	07	RLCA	дирование его с
26038	EECC	XOR #CC	помощью sub, RLCA,
26040	77	LD (HL) ,A;	xor и запись
26041	23	INC HL;	следующий адрес
26042	B7	OR A;	проверка признака
26043	ED52	SBC HL, DE;	конца, возврат в
26045	I9	ADD HL, DE;	цикл или выход
26046	20F2	JP NZ 26034;	по адресу, записан-

26048	C9	RET	ному в стеке
26049	B2EDF1		или 26049

Сначала процедура помещает в стек адрес 26049. Теперь, собственно начинается декодирование: в регистр HL заново загружается адрес 26049 - как начало закодированного блока, в DE - 27719, как адрес последнего закодированного блока. Затем, в цикл декодируются последовательно байты - инструкции SUB 34, RLCA и XOR 4CC являются ключом, с помощью которого расшифровывается эта часть программы. Наконец, проверяется условие достижения адреса 27719, как последнего декодируемого (содержится в DE). Выполняется инструкция RET, но последним записанным в стек адресом не является адрес возврата в бейсик, а записанный вначале с помощью PUSH HL адрес 26049, или адрес только что декодированного блока, следовательно, происходит его запуск.

Обратим внимание, какие инструкции осуществляют дешифрацию; никакая из них не теряет ни одного бита. Вычитание производится по модулю 256 и для разных входных данных результаты тоже различны. RLCA заменяет значения битов 7,6,3 и 2 на противоположные. Другими инструкциями, имеющими те же самые свойства являются, например, ADD, INC, DEC, RLCA, NEG, CPL, но не OR или AND. Как декодировать этот блок? Проще всего - вводя в бейсике:

POKE 26027,0 (код инструкции NOP)

Тем самым ликвидируя инструкцию PUSH HL (RET в конце программы перейдет в бейсик), и выполнить RANDOMIZE USK 26000. Стоит, однако, помнить о том, что декодирующая программа может проверяться другим фрагментом программы. В "ART studio" так и есть. Вот дальнейшая часть программы:

26283	67	LD H,A;	в A уже находится
26284	6F	LD L<A;	значение 0
26285	E5	PUSH HL	запись его в стек
26292	3AA865	LD A,(26024);	и проверка
26295	FE21	CP # 21;	содержимого ячеек
26297	C0	RET NZ;	с адресами
26298	2AA965	LD HL,(26025);	26024...26027
26301	B7	OR A;	и если оно другое,
26302	IIC165	LD DE,26049;	то стирание
26305	ED52	DEC HL,DE;	памяти с помощью
26307	C0	RET NZ;	RET NZ

26308	3AAB65	LD A, (26027);	(под адрес 0)
26311	FEE5	CP #E5;	если все нормально
26313	C0	RET NZ;	то снятие адреса 0
26314	E1	POP NZ;	и "нормальный"
26315	C9	RET	выход

Этот фрагмент проверяет: наверняка ли декодирующая процедура запустила всю программу, и если нет, то с помощью RET NZ стирает память (т.к. на стеке записи адрес 0).

Что делать в этом случае? Можно ликвидировать и эти меры защиты; но в некоторых программах это не поможет, тогда остается другой выход: снова закодировать программу, т.е. сделать обратное, чем декодирующая программа. В нашем случае следовало бы выполнить:

```
XOR #CC
RRCA
ADD 34
```

В "ART studio" неизвестно для чего была помещена кодирующая программа. Она находится под адресом 26003 и выглядит следующим образом:

26003	2IC1165	LD	HL, 26049;	адрес начала
26006	11476C	LD	DE, 27719;	адрес конца
26009	7E	LD	AA, (HL);	выборка байта из
26010	EECC	XOR	"CC";	памяти, кодирование
26012	0F	RRCA;		его с помощью XOR,
26013	C622	ADD	34;	RRCA, АДД и его
26015	77	LD	(HL), A	запись
26016	23	INC	HL;	следующий адрес
26017	B7	OR	A;	проверка окончания
26018	ED52	SCB	HL, DE;	переход в цикл или
26020	19	ADD	HL, DE;	возврат
26021	20 2	JP	HL, 26009	в бейсик
26022	C9	RET;		

Как видно, она построена аналогично декодирующей процедуре. Если такой нет, то ее можно быстро и просто написать на основе декодирующей процедуры (например ATIS ATAS, NIGHT SHADE, LAST WORD).

При защите программ применяются также малоизвестные и неопубликованные в фирменных каталогах команды микропроцессора Z80. Благодаря их применению работа программы становится малочитаемой, да и просмотр ее дисассемблером затруднен.

Наиболее часто встречаемыми неопубликованными инструкциями являются команды, оперирующие на половинках индексных регистров IX и IV в группе команд, которым не предшествует никакой иной префикс (т.е. СВН или ЕДН). Основываются они на префиксации кодом ДДН или ЕДН команды, касающейся регистра Н или L. В этом случае вместо этого регистра берется соответствующая половина индексного регистра. Через НХ обозначается старшая часть регистра XI, через IX - младшая. Аналогично НУ и IV. Вот примеры:

!	код	!	команда	!	код	!	команда	!	код	!	команда	!
!	24	!	INC Н	!	ДД24	!	INC НХ	!	FD24	!	INC НУ	!
!	2Д	!	DEC L	!	ДД2Д	!	DEC IX	!	FD2Д	!	INC IV	!
!	4C	!	LD C,Н	!	ДД4C	!	LD C,НХ	!	FD4C	!	LD U, НУ	!
!	64	!	LD Н,Н	!	ДД 64	!	LD НХ,НХ	!	FD64	!	LD НУ,НУ	!
!	260I	!	LD Н, I	!	ДД260I	!	LD НХ, I	!	FD260I	!	LD НУ, I	!
!	B5	!	DR L	!	ДДВ5	!	DR IX	!	FDВ5	!	DR IV	!

Это верно для всех команд пересылки однобайтовых между регистрами и восьмибайтовых операций AND, OR, XOR, ADD, ADC, SUB, BC и CP - выполняемых в аккумуляторе.

Префикс ЕДН или ДДН относятся ко всем регистрам Н, L или НL, присутствующим в команде. Следовательно, в одной инструкции невозможно ячейки, адресованной как (НL), регистра НL, Н или L одновременно с НХ, НУ, IX, IV (в дальнейшем ограничимся регистром IX, но все это относится также и к регистру IX), например:

66	LD	Н, (НL)	ДД66**	LD	НХ, (IX+*)
75	LD	(НL), L	ДД75**	LD	(IX+*), IX
65	LD	Н, L	ДД65	LD	НХ, IX

Несколько иначе представляется ротация ячейки, адресуемой индексным регистром, т.е. инструкций, начинающихся кодом ДДСВ. Инструкция типа RR (IT+**) и им подобные подробно описаны во всех доступных материалах о микропроцессорах 80, мало кто знает об инструкциях типа RR (IX+*), % и им подобных, где % обозначает любой регистр микропроцессора. Они основываются на префиксировании кодом ДДН или ДН инструкции типа RR %. Также обстоит дело с инструкциями SET N, (IX+*), %, а также RES N, (IX+*)% (но для BIT - уже нет). Выполнение такой

инструкции основано на выполнении нормальной команды RR (IX+*) (или подобной), SET N, (IX+*) или RES N, (IX:), а тем пересылки результата как в ячейку (IX+*), так и соответствующий внутренний регистр микропроцессора. Например:

```
СВ13      RL E
ДДСВ0113  RL (IX+I),E
```

вызывает сдвиг ячейки с адресом IX+I влево на I бит и пересылку результата в регистр E, что обычным способом следовало бы сделать так:

```
ДДСВ0116  RL (IX+I)
ДДСВ01     LD E, (IX+I)
```

В конце рассмотрения инструкции данного типа следует вспомнить, что команда EX DE, HL не дает никаких результатов, также - префиксирование команд, коды которых начинаются с EDH, а также тех, в которых не присутствует ни один из регистров H, L или пары HL (например, LD B, N, RRCA и т.д.).

Очередной любопытной командой является SLI (SHIFT LEFT AND INCREMENT), выполнение которой аналогично SLA с той разницей, что самый младший бит устанавливается в 1. Признаки устанавливаются идентично SLA и другим сдвигом:

```
СВ37      SLT A
СВ36      SLT (HL)
ДДСВ**36  SLT (IX+*)
ДДСВ**57  SLT (IX+*),A
```

Временами некоторые проблемы вызывают построение флажкового регистра, особенно тогда, когда он используется достаточно нетипично, например:

```
RUSH  AF
PDP   BC
RL    C
JP    NC,...
```

его вид представлен на рис.1:

!	7	!	6	!	5	!	4	!	3	!	2	!	I	!	0	!
!	S	!	Z	!	FS	!	H	!	F3	!	P/V	!	N	!	C	!

Дополнительной особенностью регистра E является то, что биты 3 и 5 (обозначенные как F3 и F5) точно отражают состояние восьмибитов-

вой арифметической или логической операции IN %, (C) и IN F, (C).
Например, после выполнения:

XOR A; запись значения 0
ADD A, I5; результат - 00001111

указатели будут I: F5=0; F3=

Последняя (кто поручится, что их больше нет?) тайна Z80 - это регистр обновления памяти R . А точнее то, что когда после очередного машинного цикла инкрементируется значение этого регистра, то его старший бит остается неизменным, следовательно может быть использован для хранения любой, естественно, однобитовой информации, важно то, что инструкция LD A, R , с помощью которой может быть получена эта информация, устанавливает также указатель S и, следовательно, не требуется дополнительная инструкция, проверяющая значение этого бита.

В конце несколько коротких, но часто применяемых мер защиты. Их ликвидация основывается обычно на действиях обратных защитным. Чаще всего с этой точки зрения эксплуатируются системные переменные. Например, переменная DFS7 (23659) определяет число строк в нижней части экрана, требуемое для вывода сообщения об ошибке или для ввода данных. Во время работы программы это значение обычно равно 2, но достаточно занести туда 0, чтобы программа зависла при попытке прерывания (так как туда вводится сообщение для которого нет места). Если в программе находится инструкция input или cls , то она имеет подобный результат.

Распространенным способом защиты является также изменение рамки, и атрибутов нижней части экрана. Значение отдельных битов следующее:

7	бит - FLASH
6	бит - BRIGHT
5,4,3,	биты - BORDER
2,1,0	биты - INK

Способ защиты основывается на том, что цвет чернил тот же самый, как и у рамки, следовательно, при остановке программы можно подумать, что она зависла (не видно сообщения). Разблокировать защиту можно вписав BORDER 0 или другой цвет).

Очередным простым способом защиты является изменение значения переменной ERRSP (23613-23614) или dna машинного стека (эта переменная указывает дно стека), где обычно находится адрес процедуры, обрабатывающей ошибку бейсика (вызываемой с помощью RST 8). Уменьшением значения ERRSP на 2 вызываем защиту программы от "BR" и любой другой ошибки - ошибка. Смена содержимого dna машинного стека или другая замена значения ERRSP может вызвать зависание или даже рестарт компьютера.

Орел, ул. Ленина, 1. Тип. «Труд». Зак. № 1025

Тир. 10000 экз.

ВНИМАНИЮ ТОРГУЮЩИХ ОРГАНИЗАЦИЙ И ГРАЖДАН!

МАЛОЕ ГОСУДАРСТВЕННОЕ ПРЕДПРИЯТИЕ

«АВТОМАТ» ПРЕДЛАГАЕТ:

Бытовые компьютеры (аналог «Спектрума») различной конфигурации (цветной монитор, магнитофон, джойстик, дисковод и т. д.) по ценам от 1000 до 4000 рублей.

Большой выбор игровых и обучающих программ, техническая литература — 10 книг.

Поставки осуществляются по прямым договорам.

РАДИОЛЮБИТЕЛЯМ:

Высылаем различные наборы для самостоятельной сборки компьютеров.

ЗАКАЗЫ ВЫСЫЛАЮТСЯ НАЛОЖЕННЫМ ПЛАТЕЖОМ.

НАШИ ЦЕНЫ НЕ КУСАЮТСЯ!

Адрес: 302000, г. Орел, ул. Октябрьская, 29, комн. 24.

Телефон: 6-24-53.

ПРЕДЛАГАЕТ МИИП «ПОИСК»

МИИП «Поиск» по Вашим заказам издаст газеты, буклеты, афиши, брошюры, книги, другую полиграфическую продукцию улучшенного качества, создаст согласно вашим пожеланиям рекламу и разместит ее в любых советских газетах и журналах, а также в различных зарубежных изданиях.

МИИП «Поиск» распространяет книжную продукцию повышенного спроса во многих областях европейской части СССР.

Принимаем заказы на поставку партий книг трудовым коллективам, организациям книголюбов.

Быть партнером МИИП «Поиск» выгодно для любого издательского предприятия.

Наш адрес: 302035, г. Орел, ул. Октябрьская, 35, ком. 2—15.

Телефон: 6-77-97.