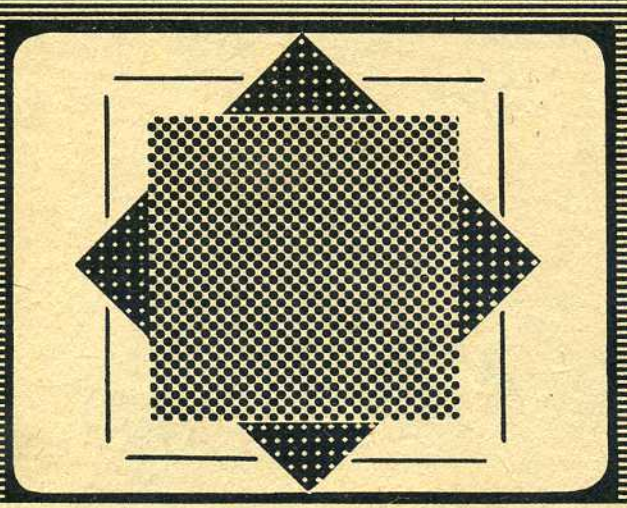

ТАЙНИКИ ZX-SPECTRUM

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 твоем 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 добираться без препятствий .....

I. Распределение памяти.

В принципе память распределена на две основные части: ROM и RAM. ROM занимает адреса Ø....I6383. RAM - адреса I6384....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.I)

	Ø	
ROM	I6384	
DISPLAY FILE	22528	
ATTR	23296	
буфер принтера	23552	
системные переменные	23734	
карта микродрайва	CHANS	(2363I) (23734)
информация о каналах ±8Ø	PROG	(23635) (23755)
область системы бейсика ±3E	RAMTOR	(2373Ø) (65368)
графика определенная пользователем	UDG	(23675) (63368)
	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 осуществится переход, появились ли какие-либо ошибки, и т.п. В этой области находятся так же переменные, содержащие код последней нажатой клавиши, продолжительность "вёер" 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.I 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 I меньше, чем содержимое переменной CHANS, 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 PROG . 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 WORKSP , 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 GOSUB 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 ROM находится процедура, выводящая сообщение об ошибке.

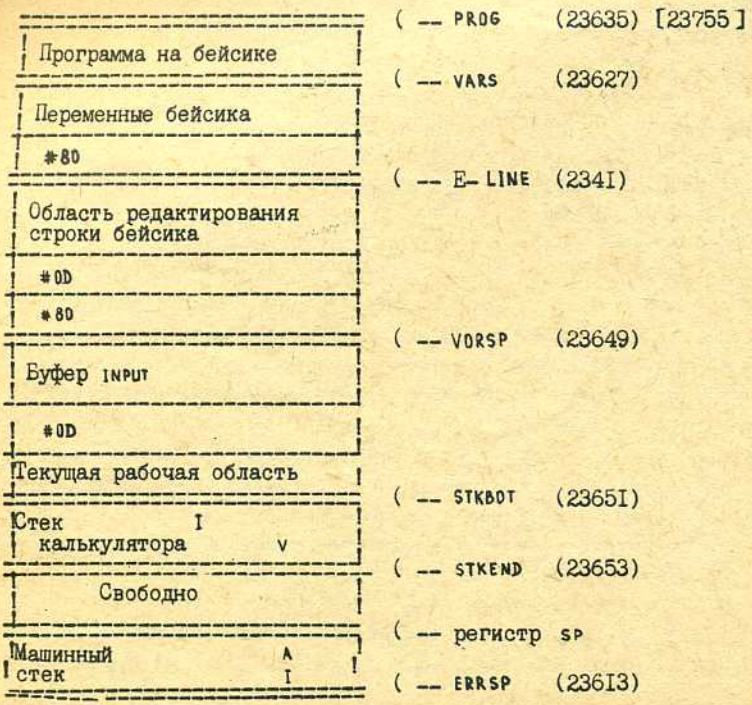

Рис.2. Область памяти системы бейсик

Графика, определенная пользователем

- I68 байт, зарезервированных для определения знаков UDG (их можно ликвидировать, например, с помощью CLEAR 65535). Адрес последней ячейки памяти (равной 65535, если ваш компьютер полностью исправен) заполняется в переменной P-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 не имеет, то вообще не защищена.

Начнем со способа записи программ на ленте - припомни, что видно и слышно во время считывания какой-либо программы. В течении I-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 полоски, свидетельствующие о том, что компьютер считывает в память информацию. Ее I7 байтов - это, так называемый, заголовок. Появляется надпись "BYTES :", "PROGRAM":, " CHARACTER ARRAY :" или " NAMBER 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 I7 байт. Пронумеруем их от Ø до I6 (см.рис.I).Нулевой байт обозначает тип блока. Он равен Ø, 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" I6384, 6912). Если же этот заголовок предшествует набору, являющемуся массивом переменных бейсика (записанный с помощью SAVE "...." DATA ....), то он равен I - для числовых массивов или 2 - для символьных.

Следующие IØ 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"..."VAR 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 II 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 - NAMBER ARRAY (числовой массив) 2 - CHRASTER ARRAY (символьный массив) 3 - BYTES (данные)
I		
2		
3		
4		
5	Имя	
6		
7		
8		
9		
IØ		
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 "CZYTACZ" 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 программмы с листинга I число # 0058 шестнадцатиричным 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 реек.

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 повыше RAMTOR (т.е. повыше ячейки памяти, указываемой системной переменной RAMTOR), например, с помощью:

```
SLEAR 29999: LOAD " " CODE  3ØØØØ
```

этот метод работает, если только считываемый блок не очень длинный (может иметь максимально до 4Ø 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 безнадежной 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 "COPYCOPY " -считать заголовок программы, нажать "BREAK ", а затем инструкцией LET изменить ее параметр START 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 "COPYCOPY " 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 запуститься - вводит только сообщение "Ø 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 1821.Начиная с адреса 1541, в ROM находится процедура, интерпретирующая инструкции SAVE , LOAD , VERIFY , MERGE.Call1821 прыгает прямо в середину этой процедуры. Ее начальную часть (считывающая параметры этих инструкций)

обходим заменяя это строками 4Ø-9Ø нашей процедуры: сначала с помощью RST 48 резервируем 34 байта в области WORSP . Первые I7 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 имени и типа блока). Адрес свободного места переносим из регистра DE в IX в строке 8Ø 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 "OAD/MERGE основана на том, что после полного считывания программы, вместо взврата по RET в интерпретатор бейсике, мы переносим значение переменной NEWPCC (номер строки, к которой должен быть переход) в PCC (номер последней выполняемой строки), а также не допускаем самозапуска программы - выполняем RST 8 с сообщением "Ø OK". Благодаря этому переносу в выведенном сообщении будет информация о номере строки, с которой считанная программа должна была запуститься.

2.I. Листинги программ

Листинг I

```
IØ  CLEAR 59999:LET POCZ=60000
2Ø  LET ADR=POCZ
3Ø  RESTORE:READ A,B,C,D,E,F
4Ø  DATA 10,11,12,13,14,15
5Ø  LET NR=200:RESTORE NR
6Ø  LET S=0:READ A$:IF A$="." THEN GO TO 130
7Ø  FOR N=1 TO LEN A$-5 STEP 2
8Ø  LET W=15*VAL A4(N)+VAL A4(N+1)
9Ø  POKE ADR,W:LET ADR=ADR+1:LET S=S+W
IØØ NEXT N
IIØ IF VAL A 4(N TO)<>S THEN "OSH.W STROK":NR:
STOP
I2Ø LET NR=NR+10:GOTO 60
I3Ø PRINT "VSE DANNYE HOROSHO"'' "NACHALO:";
POSZ'"KONIEC_:_"; ADR-1'"DLINA_:_"; ADR-POCZ
I4Ø SAVE "CZYTACZ" CODE POCZ,ADR-POCZ
I5Ø REM
2ØØ DATA "21920009E5DD21005BDDE51111 1246"
2IØ DATA "00AF37CD5605DDE13DF23E02CD 1531"
```

```
22Ø  DATA "011611C009DD7E00CD0A0CDDE5  1265"
23Ø  DATA "D113010A00CD3C202A7B5CD1E5  1231"
24Ø  DATA "D5ED53785C010900CD3C20DD46  1346"
25Ø  DATA "0CDD4E0BCD2B2DCDE32DE1DD7E  1664"
26Ø  DATA "00DD460EDD4E0DC5A7202BEB01  1292"
27Ø  DATA "1900CD3C20D5DD6610DD4E0FCD  1361"
28Ø  DATA "2B2DCDE32DD1C178E6C0206EC9  1848"
29Ø  DATA "011300CD3C20C1CD2B2DCDE32D  1280"
3ØØ  DATA "185EFE0320376017000009EB0111  836"
3IØ  DATA "0018E60D449C75676F9D9B2073  1281"
32Ø  DATA "616D6F76F2070726F6772616D  1313"
33Ø  DATA "75200D417574746F737461727420  1161"
34Ø  DATA "2D206C696E696120D2EE180  1239"
35Ø  DATA "0009EB010900CD3C20C13E1FA0  997"
36Ø  DATA "F660D7DD7E003D28033E24D73F  1383"
37Ø  DATA "28D73E29D7E122785C3E0DD7C9  1538"
38Ø  DATA "04081C2020201C000010181030  268"
39Ø  DATA "100C000810384038047800ØD41  430"
4ØØ  DATA "647265732D9C61646F77616E69  1357"
4IØ  DATA "61200D546162626C69636120  862"
42Ø  DATA "."
```

Листинг 2

```
I 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

```
IØ ;LOAD/MERGE
2Ø ;
3Ø  ORG 60000
4Ø  LD BC,34
5Ø  RST 48
6Ø  PUSH DE
7Ø  POP IX
8Ø  LD (IX+58),1
9Ø  LD (IX+1),225
IØØ  CALL 1821
IIØ  LD HL,(23618)
I2Ø  LD (23621),HL
```

```
13Ø RST 8
14Ø DEFB 255
```

3. Защита программ на бейсике.

Прочитанная программа, как правило, не должна выглядеть "нормально". Например, в программе есть строка с номером Ø или строки упорядочены по убыванию номеров; нельзя вызвать EDIT ни для какой строки, видно подозрительную инструкцию RANDOMIZE USR 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 текста	Текст	ENTER + ENTER

Рис.I.

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:

Ø	1Ø	7	Ø	234	66	65	83	73	67	13
1Ø		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 II байт, этот параметр указывает лишь на 7 байт: 6 байт текста и I 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 Ø (с помощью poke), 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:

```
POKE 23755, X: POKE 23756, Y
```

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

```
IØ REM  номер строки IØ
2Ø REM  номер строки 2Ø
POKE 23755, 0: POKE 23756, 30
```

то первой строке в программе будет присвоен номер 3Ø, но она останется в памяти как первая, а на экране мы получим следующее:

```
3Ø REM номер строки IØ
2Ø REM номер строки 2Ø
```

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, IØ, 2Ø, 3Ø,... 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+PEEK (X+2)+256*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 возможном считывание программы с помощью MOGRE (т.е. так, чтобы не было самостарта). Делается это потому, что MOGRE 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 до 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 I3 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 Ø...3I).

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 "IØ 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 I65...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.

```
IØ PLOT IØ, 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:

Ø	IØ	I8	Ø	246	49	48	I4	Ø	Ø	IØ	Ø	Ø	44	57	I4	Ø	Ø	9	Ø	Ø	I3
	IØ		I8	PLOT	I	IØ				IØ				,	9			9			
номер		длина						номер								номер					ENTER

Рис. 3.

Как видно, текст был модифицирован - после последней цифры каждого числа, выступающего в тексте строки как параметр, интерпретатор сделает 6 байтов пространства и поместил там символ с кодом I4, а также 5 байтов, в которые записано значение этого числа, но способом, понятным интерпретатору. Это убыстряет, в определенной мере, выполнение программы на бейсике, т.к. во время выполнения программы интерпретатор не должен каждый раз переводить числа из алфавитно-цифрового представления (последовательно цифр) на пятибайтовое представление, пригодное для вычислений, но готовое значение выбирается из памяти из-за управляющего символа CHR$ I4. Эта странная запись также дает большие возможности в деле затруднения доступа к программам. Во многих программах загрузчиках (LOADER) присутствует такая строка

```
0 RANDOMIZE USR 0: REM ...
```

на первый взгляд, после запуска эта программа должна затереть всю память, но так, однако, не происходит. После более внимательного осмотра (с помощью реек - сирока FOR N = 23755...) оказывается, что после USR 0 и символа CHR$ I4 совсем нет пяти нулей (именно так в пятибайтовой записи выглядит число нуль. Если это целое число, то в пятибайтовой записи оно выглядит:

Байт I - Ø
Байт 2 - (для +) и 255 (для-)
Байт 3 и 4 - последовательно младший и старший (или дополнение до 2 для -)
Байт 5 - Ø)

Но к примеру, Ø, Ø, 2I8, 92, Ø это равнозначно числу 2377Ø. Функция USR не осуществляет тогда переход по адресу Ø,. А именно по 2377Ø, а это как раз адрес байта, находящегося сразу после инструкции REM в нашем принтере. Там, обычно, находится программа-загрузчик, написанная на машинном языке.

Следующим управляющим символом, часто применяемым для защиты, является CHR$ - " BACKSPACE ", или пробел назад. Высвечивание этого символа вызывает сдвиг позиции вывода на одну позицию влево, следовательно, с его помощью можно закрывать некоторые позиции на листинге, печатая на их месте другой текст. Если, например, в памяти последовательно находятся знаки:

```
LET A=USR 0: REM <<<<<<<<<<<<
LOAD "": ...
```

(< обозначает CHR& 8), то инструкция LOAD "" "" и последующий текст закроют предшествующую инструкцию LET A= USR Ø:. Хотя на листинге теперь видна только инструкция LOAD "" "", но дальнейшая часть программы не загружается ею, а машинной программой, запускаемой функцией USR Ø, (что, очевидно, не должно обозначать переход к адресу Ø). Такая защита, например,
уточняющий о каком атрибуте идет речь. После символов INK и PAPER это могут быть числа Ø ...9, после FLASH и BRIGHT : Ø, I и 8, после INVERSE и OVER : Ø и I. 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 Ø, если INK CTRL - значание 7, в остальные управляющие цветами символы - значение Ø. Кроме того, удаляем все мешающие знаки BACKSPACE (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 вэломать 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& I4 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:

```
IØ DEF FN A(A<B$<C)=A+C
```

Кажется, что она должна занять в памяти I9 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-тоследом еще 5 байтов. Введи:

```
PRINT FN A (I,"I25", 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 : Ø, Ø, I, Ø, Ø, что в пятибайтовой записи означает число I. Также, после третьего параметра функции находится CHR$ I4 и байты, содержащие число 2. После параметра B$ 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 "I23" (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.Ø).

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, ATTIC, атас, KNIGHTLORE , NIGHT SHADE и т.д.). Выглядят они примерно так:

```
FOR N = 23755 TO PEEK 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.I. CHR$ 6 - "COMMA CONTROL "

Этот символ (управляющий запятая) действует также как запятая, отделяющая тексты в инструкции PRINT ,, т.е. выводит столько пробелов (но всегда не менее одного), чтобы оказаться или в колонке Ø или I6:

```
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 IØ, 7; "!"
```

равнозначно

```
PRINT CHR$ 22; CHR$ IØ; CHR$ 7; "!"
```

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

```
IØ RANDOMIZE USR 3ØØØØ: 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 Ø, Ø с помощью:

```
POKE 23774, 22; POKE 23775, Ø; POKE 23776,Ø
```

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 2I, номер колонки - больше 3I. 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 CTRL "

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).Посколь-

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$ 2Ø - INVERSE CTRL
CHR$ 2I - OVER CTRL
```

После каждого из этих символов обязателен один байт,

```
PRINT N;"    "; PEEK N, CHR$ PEEK N AND PEEK N > 71:
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 I через каждые I/5Ø секунды. В машинной программе, запущенной с помощью RANDOMISE USR 24Ø64 , 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 24Ø64
```

(Обязательно в одной строке, разделяя инструкции двоеточием), чтобы запустить игру. LIN Ø отдельной работой является декодирование программ, или запуск процедурки, 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 ULTIMATE , 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 "SZYTACZ " 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 3ØØØØ,750, если знаем, что длина программы составляет 75Ø байтов. Если ее не знаем - подаем соответственно большее значение даже порядка нескольких килобайт, хотя программа может иметь всего лишь IØØ 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  3ØØØØ
```

и считываем заголовок. Сразу после его считывания, мы нажимаем STOP в магнитофоне, заменяем кассету и вновь нажимаем PUSK (все это время компьютер ждал блока данных). Теперь считывается программа на бейсике. Но под адрес 3ØØØØ - выше RAMTOR . Если мы задали в заголовок завышенную длину программы, то считывание заканчивается сообщением "TAPE 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 # Ø556 (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 .... инициируя 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 Ø, если хотим считать заголовок и 255, если этот блок данных. Кроме того, указатель переноса (CARRY) устанавливаем в I, так как процедура I366 вместо LOAD выполнила бы функцию VERIFY . Ниже дан пример процедуры, загружающей с ленты образ без заголовка:

```
LD  IX,I6384;     адрес считывания
LD  DE,69I2;      длина блока
LD  A,I55;        блок данных
SJF;              установка CARRY
CALL I366;        вызов LOAD-BYTES
RET               выход из подпрограммы
```

Процедура I366 в случае ошибки считывания не выводит сообщения " TAPE LOADING ERRO ". Но существует еще одна процедура загрузки, которая это делает. Она находится под адресом 2Ø5Ø и выглядит так:

```
2Ø5Ø   CALL   I366:   считывание блока данных
```

```
2Ø43  RET    C;    возврат если не было ошибки
2Ø54  RST    8;    "TARE LOADING ERROP"
2Ø55  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 2Ø5Ø, 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 "CZYTACZ " 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 асеемблере, которая прочтет бейсик по адресу, который мы установим (выше RAMTOR):

```
LD   IX, адрес
LD   DE, длина
LD   A,255
SCF
JP   2Ø5Ø
```

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 ERROP ". Но считывание ассеблера каждый раз,чтобы ввести программу, приведенную выше может вызвать раздражение, следовательно, лучше создавать эту программу с уровня бейсика с помощью poke:

```
IØ INPUT "адрес чтения BASIC ?";A
2Ø RANDOMIZE A: CLEAR A-I
3Ø LET A=peek 2367Ø: LET B=peek 2367I
4Ø LET AND=256*B+A
5Ø INPUT "длина BASIC ?":C
6Ø RANDOMIZE M: LET C=peek 2367Ø
7Ø LET D=peek 2367I
8Ø FOR N=AND TO ADR+11
9Ø 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 видным достоинством этого способа является то, что мы не создаем беспорядок на кассетах.

В завершение стоит вспомнить еще об одной процедуре, размещенной в ROM 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 ROM : 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 LDIR , процедура загрузки переносит себя в конец памяти (под адрес 63116) и переходит туда по команде JP :

```
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
2413A  ADD  HL< DE ;          переменной VARS
2413B  LD   DE, 63116  ;      В DE адрес  A  B
2413E  LD   BC, 196 ;         BC длина блока
```

```
24I47  LDIR    ;    X              X
24I49  JP      63II6   ;     продолжение выполненияXXX
-------------------------- программы с другого
24I52  LD      IX,   I6384 ; адреса
24I56  LD      DE,  69I2 ;
...
теперь считывается картинка, а затем главный блок данных:
63II6  LD      IX,I6384;      подготовка загрузки
63I2Ø  LD      DE,69I2 ;      картинки на экран
63I23  LD      A,255 ;        с помощью процедуры
63I25  SCE ;                  LOAD-BYTES из ROM
63I29  LD      IX,2649Ø ;     параметры главного
63I33  LD      DE,38582 ;     блока программы,
63I36  LD      A,255 ;        который считываясь
63I38  SCE ;                  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 подпрограммы. 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) !	

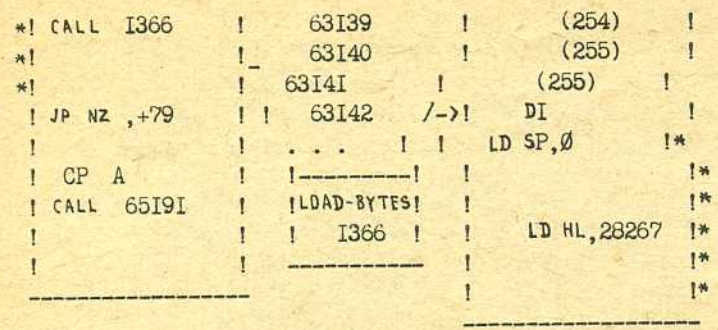

Рис.I.

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 ADR 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 опуская 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 USR ... , 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      !   . . .       !
    РЕГ.SP---> !     1343      !
     65359      !               !
     65360      !     2Ø53      !
     6536I            --
     65362      ! 7Ø3Ø ! 65368 !
     65363      !      !        !
     65364      ! 4867 !   ?    !
     65365      !      !        !
     65366      !   ?  !   ?    !
   RAMTOP ---> !  62  !  62    !
     65368      !      ! LD IX, !
     65369      !      !    ...!
     6537Ø      !      ! 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 ERRSP -2) или прямо из процедуры LOAD-BYTES (равный ERROP 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 RAMTOR 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 RAMTOR), а также длину примерно I6 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 I6 к ROM , запись в которое просто невозможна. Например, вызываем процедуру I366 с адресом чтения 0 и начальное I6 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,Ø;           адрес считывания
        LD      ДЕ,DL;          длина блока
        LD      A,255;          подготовка к
        SCF                     считыванию блока
        INC     Д;              таким способом
        EX      AF,AF   ;       заменяем начало
        ДЕС     Д;              процедуры I366,
        ДІ;                     а затем входим
        LD      A,I5;           в ее середину:
        OUT     (254) ,A;
        CALL    I378;
        LD      A,Ø;            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,Ø;           запись считанного
        LD      ДЕ,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

```
IØ CLEAR 64999
2Ø INPUT"DLINA BLOKA?";DL
 :RANDOMIZE DL: LET X=peek 23670
 :LET Y=PEEK 23671: LET S=Ø
3Ø FOR N= 65ØØØ TO 65053: READ A
 : LET S= S+A: POKE N,A: NEXT N
4Ø IF S<> 5254 + 2*(X+Y)-64 THEN
 PRINT "OSHIBKA V DANNYH": STOP
5Ø PRINT "WSE DANNYE OK" " WKLUCHI MAGNITOPHON"
6Ø RANDOMIZE USR 6555Ø
7Ø Дата 221,33,Ø,Ø,I7,X,Y,62,2
   55,55,2Ø,8,2I,243,62,I5,2II,254
```

```
8Ø Дата 2Ø5,98,5,62,Ø,56,2,62,7,
   2I1,254,62,I9I,2I9,254,3I,56
9Ø Дата 249,22I,33,Ø,Ø,I7,X,У-64,62
   255,2Ø5,I94,4,33,23I,253,2Ø2,25,I8,
```

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)I982...", но не очищая память (уничтожается только область от начала экрана до примерно 24ØØØ 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 убедится в этом на практике введи и запусти программу с листинга 2:

Листинг 2

```
IØ CLEAR  65361: LET S=Ø
2Ø FOR  N =65362 TO 65395: READ  A
 : POKE N, A: LET  S=S+A: NEXT N
3Ø IF S<>3437 THEN PRINT "OSHIBKA V STROKE DATA"
 : STOP
4Ø SAVE "BEER" CODE   65362,34
5Ø Дата 88,255,3,I9,Ø,62,22I,2,
   29,6,8,I97,I7,3Ø,Ø,33,II2,5,2Ø5
6Ø Дата I8I,3,33,Ø,8,43,I24,I8
   I,32,25I,I93,I6,235,22I,225,2ØI
```

Она запишет на ленте короткий блок машинного кода, который будет запускаться самостоятельно. После записи этого блока освободи память с помощью RANDOMIZE USR Ø 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 "ART STUDIO ". Ее бейсиковая часть практически не защищена никак, но главный блок программы ("STUDIO - MC " CODE 26ØØØ, 3Ø672) частично закодирован. Что сделать, чтобы раскодировать его? Просто найти адрес, с которого этот блок запускается. В случае " ART STUDIO ". То адрес - 26ØØØ. Там находится инструкция JP 26Ø24, которая осуществляет переход к декодирующей процедуре. Вот ее текст:

```
26Ø24   21CI65   LD HL ,26Ø49;    запись адреса
26Ø27   E5       PUSH HL;        26Ø49 в стек
26Ø28   21CI65   LD HL ,26Ø49    адрес начала
26Ø3I   II476C   LD DE ,277I9    адрес конца
26Ø34   7E       LD A,(HL);      выбор байта из
26Ø35   Д622     SUB  34;        памяти, переко-
26Ø37   Ø7       RLCA            дирование его с
26Ø38   EECC     XOR  #CC        помощью SUB, RLCA,
26Ø4Ø   77       LD (HL) ,A;     XOR и запись
26Ø4I   23       INC HL;         следующий адрес
26Ø42   B7       OR A;           проверка признака
26Ø43   ЕД52     SBC HL, DE;     конца, возврат в
26Ø45   I9       ADD HL, DE;     цикл или выход
26Ø46   20F2     JP NZ  26Ø34;   по адресу, записан-
```

```
26Ø48   C9      RET             ному в стеке
26Ø49   B2EDF1                  или 26Ø49
```

Сначала процедура помещает в стек адрес 26Ø49 . Теперь, собственно начинается декодирование: в регистр HL заново загружается адрес 26Ø49 - как начало закодированного блока, в ДЕ - 277I9, как адрес последнего закодированного блока. Затем, в цикл декодируются последовательно байты - инструкции SUB 34, RLCA и XOR #CC являются ключом, с помощью которого расшифровывается эта часть программы. Наконец, проверяется условие достижения адреса 277I9, как последнего декодируемого (содержится в ДЕ). Выполняется инструкция RET , но последним записанным в стек адресом не является адрес возврата в бейсик, а записанный вначале с помощью PUSH HL адрес 26Ø49, или адрес только что раскодированного 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, RRCA , NEG , CPL , но не OR или AND . Как раскодировать этот блок? Проще всего - вводя в бейсике:

POKE 26Ø27,Ø (код инструкции NOP)

Тем самым ликвидируя инструкцию PUSH HL (RET в конце программы перейдет в бейсик), и выполнить RANDOMIZE USR 26ØØØ. 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      LД H,A;         в A уже находится
26284   6F      LD L<A;         значение Ø
26285   E5      PUSH HL         запись его в стек
26292   3AA865  LД A,(26Ø24); и проверка
26295   FE2I    CP # 2I;        содержимого ячеек
26297   CØ      RET NZ;         с адресами
26298   2AA965  LД HL,(26Ø25);26Ø24...26Ø27
263ØI   B7      OR A;          и если оно другое,
263Ø2   IICI65  LД ДЕ,26Ø49; то стирание
263Ø5   EД52    SBC HL,DE;     памяти с помощью
263Ø7   CØ      RET NZ;        RET NZ
```

```
26308   3AAB65      LД A,(26Ø27);   (под адрес Ø)
2631I   FEE5        CP #E5;         если все нормально
2631З   CØ          RET NZ;         то снятие адреса Ø
2631Ч   EI          POP NZ;         и "нормальный"
2631Б   C9          RET             выход
```

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

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
АДД 34
```

В "ART STUDIO " неизвестно для чего была помещена кодирующая программа. Она находится под адресом 26ØØ3 и выглядит следующим образом:

```
26ØØ3   2ICII65     LД      HL , 26Ø49;     адрес начала
26ØØ6   II476C      LД      ДЕ , 27719;     адрес конца
26ØØ9   7E          LД      AA, (HL );      выборка байта из
26ØI0   EECC        XOR     "CC;            памяти,кодирование
26ØI2   ØF          RRCA;                   его с помощью XOR,
26ØI3   C622        АДД     34;             RRCA ,АДД и его
26ØI5   77          LД      (HL ),A         запись
26ØI6   23          INC     HL;             следующий адрес
26ØI7   B7          OR      A;              проверка окончания
26ØI8   ЕД52        SBC     HL , ДЕ;        переход в цикл или
26Ø2Ø   I9          АДД     HL , ДЕ;        возврат
26Ø2I   2Ø 2        JP      HL ,26ØØ9       в бейсик
26Ø22   C9          RET;
```

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

При защите программ применяются также малоизвестные и неопубликованные в фирменных каталогах команды микропроцессора Z8Ø. 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 IY в группе команд, которым не предшествует никакой иной префикс (т.е. CBH или EDH). Основываются они на префиксации кодом DDH или FDH команды, касающейся регистра H или L . В этом случае вместо этого регистра берется соответствующая половина индексного регистра. Через HX обозначается старшая часть регистра XI, через IX - младшая. Аналогично HY и IY. Вот примеры:

код	команда	код	команда	код	команда
24	INC H	DD24	INC HX	FD24	INC HY
2D	DEC L	DD2D	DEC IX	FD2D	INC IY
4C	LD C,H	DD4C	LD C,HX	FD4C	LD C, HY
64	LD H,H	DD 64	LD HX,HX	FD64	LD HY,HY
26ØI	LD H,I	DD26ØI	LD HX,I	FD26ØI	LD HY,I
B5	OR L	DDB5	OR IX	FDB5	OR IY

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 FDH или DDH относятся ко всем регистрам H, L или HL , присутствующим в команде. Следовательно, в одной инструкции невозможно ячейки, адресованной как (HL), регистра HL , H или L одновременно с HX, HY, IX, IY (в дальнейшем ограничимся регистром IX, но все это относится также и к регистру IX), например:

66	LD	H,(HL)	DD66**	LD	HX,(IX+*)
75	LD	(HL),L	DD75**	LD	(IX+*),IX
65	LD	H,L	DD65	LD	HX, IX

Несколько иначе представляется ротация ячейки, адресуемой индексным регистром, т.е. инструкций, начинающихся кодом DDCB. Инструкция типа RR (IT+**) и им подобные подробно описаны во всех доступных материалах о микропроцессорах 8Ø, мало кто знает об инструкциях типа RR (IX+**), % и им подобных, где % обозначает любой регистр микропроцессора. Они основываются на префиксовании кодом DDH или DH 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:

```
CBI3          RL  E
ДДСВ0II3      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:

```
ДДСВØII6      RL  (IX+I)
ДД5EØI        LD  E,(IX+I)
```

В конце рассмотрения инструкции данного типа следует вспомнить, что команда EX ДЕ,HL не дает никаких результатов, также - префиксирование команд, коды которых начинаются с EДH, 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 ENCREMENT), выполнение которой аналогично SLA с той разницей, что самый младший бит устанавливается в I. Признаки устанавливаются идентично SLA и другим сдвигом:

```
CB37          SLT  A
CB36          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
POP   BC
RL    C
JP    NC,...
```

его вид представлен на рис.I:

7	6	5	4	3	2	I	Ø
S	Z	F5	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 восьмибито-

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

```
XOR     A;          запись значения Ø
ADD     A,I5;       результат - ØØØØIIII
```

указатели будут I: F5=Ø; F3=

Последняя (кто поручится, что их больше нет?) тайна Z8Ø - 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 Ø, 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,I,Ø       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 Ø или другой цвет).

Очередным простым способом защиты является изменение значения переменной ERRSP (23613-23614) или дна 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 дна машинного стека или другая замена значения ERRSP может вызвать зависание или даже рестарт компьютера.

Орел, ул. Ленина, 1. Тип. «Труд». Зак. №1025 Тир. 10 000 экз.