
КАК РАБОТАТЬ
С ЗАЩИЩЕННЫМИ ПРОГРАММАМИ
(ТАЙНИКИ СПЕКТРУМА)

К А К Р А Б О Т А Т Ь

С З А Щ И Щ Е Н Н Ы М И П Р О Г Р А М М А М И

(ТАЙНИКИ СПЕКТРУМА)

учебно-методическое пособие
для пользователей ПЭВМ ZX - СПЕКТРУМ

НАУЧНО-ПРОИЗВОДСТВЕННЫЙ ЦЕНТР "ТИНОПОС"

НОВОПОЛОЦК

1991 г

ОГЛАВЛЕНИЕ

Копирование программ, особенно компьютерных игр, весьма хитрая процедура из-за того, что ряд программ имеет очень сложную структуру, состоит из большого количества разнотипных файлов, не всегда отвечающих стандартам ОС. Ситуация осложняется большими размерами файлов, занимающими иногда все ОЗУ без остатка, включая экран и область системных переменных ОС, что приводит к ее блокированию. Если вы имеете дело с чужой (фирменной) программой, то загрузив ее, вам не так просто ее выгрузить. Во-первых, она автостартует и остановить ее очень сложно, во-вторых, вам неизвестны адреса блоков, из которых она состоит. Однако все эти проблемы можно решить с помощью специальных программ копировщиков, которых разработано очень много. Но и копировщик может оказаться бессилен, если в копируемой программе предусмотрены специальные, и порой весьма изощренные, меры защиты от копирования. Лля преодоления таких препятствий разработаны специальные программы копировщики, которые позволяют копировать защищенные программы. Но даже если удается скопировать программу, далеко не всякую программу можно прочитать, внести в нее изменения, так как помимо защиты от копирования во многих фирменных программах предусмотрены меры защиты программ от их остановки и модификации. Ваша попытка вмешаться в программу может привести к самым разным последствиям - от зависания компьютера до полного обнуления его памяти - в зависимости от того, какое "наказание" придумал автор зашиты.

Рассмотрению методов, позволяющих преодолеть встроенные в программы средства зашиты и все-таки прочитать программу, а при необходимости и модифицировать ее, и посвящено это пособие.

1. ОБЩИЕ СВЕДЕНИЯ О СПОСОБАХ ЗАШИТЫ ПРОГРАММ

Напомним вкратце, что происходит при нормальном считывании программы с ленты в компьютер. Перед загрузкой программы убедимся, что память компьютера очищена.

Обычно СПЕКТРУМ выполняет SAVE и LOAD с одной постоянной скоростью порядка 1500 бод (бит/сек)

В течение первых 1-2 секунд считывания на экране видны широкие красные и синие полосы, а также слышен длительный звук. Это так называемый пилот (или пилоттон, или просто тон) , который позволяет компьютеру синхронизироваться с сигналом ленты, которую он будет считывать и обеспечивает надежность ввода кодов в память компьютера. Затем появляются мерцающие тонкие желтые и синие полоски, свидетельствующие о том, что компьютер считывает в память байты информации. которые могут быть и бейсиком, и машинными кодами, и данными. Пилот вместе с последующими байтами называется блоком кодов

При считывании отдельных программ вместо плавного

перемещения по экрану красных и синих полос наблюдается резкое (рывками) их перемещение по экрану, сопровождающееся отрывистыми звуковыми сигналами. Это прерывистый пилот (или джеркитон), разработанный для защиты программ от копирования. Его часто применяют такие фирмы как "ULTINAME" и "OCEAN".

Иногда применяют более высокие скорости считывания информации, при этом ширина желто - синих полос на экране меньше, а тон звука, сопровождающего чтение, выше. Надежность считывания информации при таком способе зашиты, конечно, ниже.

Ряд фирм для зашиты своих программ от копирования разработали свои пилоты. Например может применяться значительно более узкий пилот (порядка 0,1 сек. вместо обычных 1-2 секунд), или наоборот более широкий пилот. При этом бордюрные линии становятся широкими желто-синими. Это так - называемый ТОНИНГ -метод.

Можно записать программу без зазоров между блоками кодов, при этом при прослушивании записи слышен один непрерывный Тон, а полосы на экране будут широкими желто - синими. Непрерывный тон будет сбивать копирующие программы,поскольку они будут ждать перерыва.

Могут задаваться блоки избыточной длины (более 50 К), или блоки длиной 17 байтов (так называемые фальшхэдеры), которые воспринимаются копировщиком как заголовок очередного блока, в то время как они таковыми не являются.

Могут загружаться блоки с замеряемой паузой между ними.

Одни методы защиты могут применяться в комбинации с другими, что повышает действенность защиты. Обычно принцип их действия состоит в том, что первый блок программы является бейсик-загрузчиком, которых подготавливает загрузку второго блока. Второй блок в машинных кодах готовит нестандартную загрузку прочих блоков, которые уже не могут быть никуда загружены, где предварительно не отработал второй блок, в том числе и в копировщик.

Лля копирования защищенных программ разработан ряд специальных программ копировщиков (например LERM 7), которые со многими защитами справляются.

Если программа скопирована и считана в память компьютера, то следующим препятствием для ее прочтения выступают встроенные в программу защиты, которые не позволяют остановить программу, "пролистать" ее, кодируют программу, тем или иным способом делают ее текст нечитаемым, или просто стирающие DOB память компьютера при любой попытке вмешательства в программу. 6 числе основных направлении, по которым идет разработка методов защиты программ. можно назвать следующие:

- исключение возможности остановки программ:
- исключение возможности подачи команды LIST;
- сделать листинг программы нечитаемым;
- кодирование программ специальными кодирующими процедурами.

Приступим к подробному рассмотрению наиболее распространенных способов зашиты программ.

2. ОРГАНИЗАЦИЯ ПАМЯТИ КОМПЬЮТЕРА

. Для обеспечения работы операционной системы е памяти компьютера выделено несколько областей. Эти области отделены друг от друга некоторыми границами, которые могут быть фиксированы (постоянны), а могут изменяться в зависимости от конкретных условий. Фиксированные границы обозначены числом, а переменные - именем системной переменной, в которой они хранятся. Распределение памяти приведено на рисунке 1.

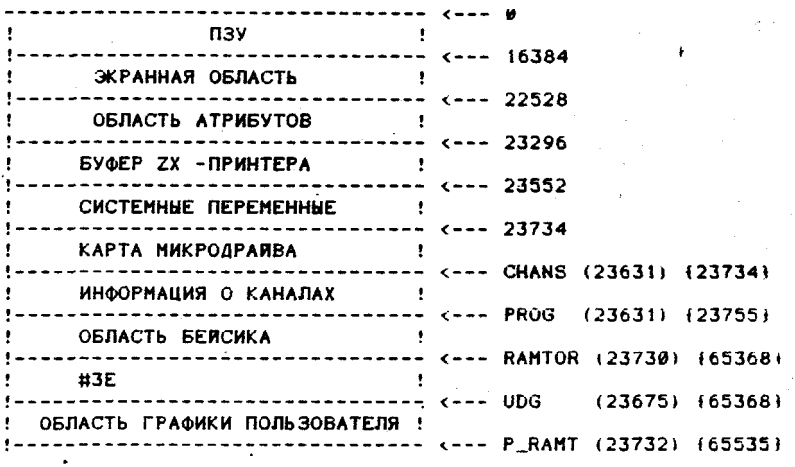


рисунок 1.

примечание: на рисунках 1 и 2 число за стрелкой означает адрес начала указываемого блока, если адрес перемещаемый - то вместо числа записано имя системной переменной, содержащем этот адрес, а в скобках - адрес этой системной переменной, в фигурных скобках - значение этой системной переменной после включения компьютера (или выполнения RESET), но без подключения внешних устройств.

Адреса с 0 по 16383 - это область ПЗУ, где хранятся:
7 кбайт - операционная система (ОС)
8 кбайт - интерпретатор бейсика
1 кбайт - генератор знаков

Адреса с 16384 по 22527 - экранная область - обеспечивает высокое разрешение экрана телевизионного приемника (ТП). достигающееся за счет того, что в отведенных под экранную область 6К памяти можно закодировать изображение любой из 49192 точек экрана ТП (матрица 192x256 точек)

Экранная область разбита на три части по 2К:
- строкам в-7 соответствует адреса 16384-18431;
- строкам 8-13 соответствуют адреса 18432-2в479;
- строкам 16-23 соответствуют адреса 20480-22527.
Каждые из этих третей по 2К состоят из 8 массивов по 1/4 кбайта. Каждый массив из 1/4 кбайта (256 байтов) хранит коды символов одной строки экрана. Счет линий начинается сверху экрана.

Адреса с 22328 по 23295 - ОБЛАСТЬ АТРИБУТОВ.

На экране может быть адресовано 768 символов, для каждого из которых может быть задан один из восьми цветов "чернил", один из восьми цветов "бумаги", признак мигания и признак яркости - повышенной или нормальной.
Область атрибутов имеет 768 байт - по 1 байту на каждую позицию символа, в которых хранятся признаки цвета, яркости, мигания.
Кодировка байтов атрибутов следующая:
биты 0-2 - код цвета "чернил"
биты 3-5 - код цвета "бумаги" (фон)
бит 6 - признак яркости (0 - обычная, 1 - повышенная)
бит 7 - признак мигания (0 - постоянное свечение,
1 - мигание символа)
Коды цветов следующие:
0 - черный; 1 - синий; 2 - красный; 3 - фиолетовый;
4 - зеленый; 5 - голубой; 6 - желтый; 7 - белый.

Адреса с 23296 по 23551 - ОБЛАСТЬ БУФЕРА ZX-ПРИНТЕРА.

Эта область (от 23296 до 23551) используется лишь во время работы компьютера с принтером. Если не используются инструкции. касающиеся принтера (LLIST, LPRINT, COPY), то его содержимое не меняется и его можно использовать для других целей. Необходимо помнить, что использование какой-либо из этих инструкций даже без подключения принтера, произведет в этой области изменения.

СИСТЕМНЫЕ ПЕРЕМЕННЫЕ - адреса с 23552 по 23733 (128 байтов).
Эти ячейки памяти используются системой для запоминания необходимых для его безошибочной работы данных, таких, как например, адреса так называемых подвижных блоков памяти, информация о выполнении программы в бейсике, т.е. какая строка выполняется, к которой должен осуществиться переход, появились ли какие-нибудь ошибки и т.п. В этой области находятся переменные, содержащие код последней нажатой клавиши, продолжительность "BEEP" клавиатуры и еще много других. Непосредственно за системными переменными, которые кончаются адресом 23733, начинаются так называемые подвижные области памяти. Это означает, что адреса их начал и концов (а также длины) могут изменяться в зависимости от того, подклинены ли какие-нибудь внешние устройства, какова длина программы на бейсике. сколько образуется переменных.

КАРТА МИКРОДРАИВА адреса с 23734 до CHANS-1
Если к компьютеру подключен ZX ИНТЕРФЕЙС 1, то с адреса 23734 до адреса на 1 меньше, чем содержимое переменной CHANS, находится карта микродрайва-область,используемая как буфер для трансляции данных, как набор добавочных системных переменных и т.п. 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), определяющего конец этого блока (это так называемый указатель конца блока)
ОБЛАСТЬ БЕЙСИКА
Распределение области бейсика приведено на рисунке 2.

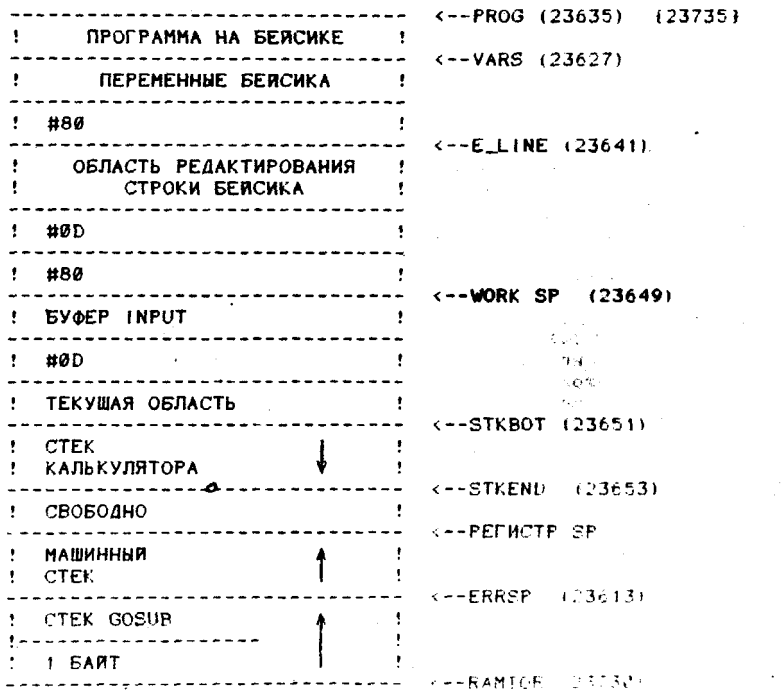


РИСУНОК 2

Эта область памяти содержит текст введенной программы на бейсике. Адрес 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 содержащегося в переменной E_LINE, находится область, используемая во время редактирования строки бейсика, а также ввода директив с клавиатуры (т. е. когда в нижней части 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 WORK SP, находится область буфера INPUT, завершаемая 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 лент программ. Туда считывается программа, размещаемая в памяти с помощью MERGE, прежде чем будет присоединена к уже существующей программе. Эта область используется тогда, когда требуется на определенное время немного свободной памяти, но не только во временное пользование.

Однако, системе бейсика принадлежит область памяти до ячейки указываемой системной переменной RAMTOR. По этому адресу находится число #3E (62). которое устанавливает конец используемой бейсиком области. Двигаясь по памяти "вниз", мы сталкиваемся с одним не используемым байтом,который является как бы младшим байтом двухбайтового числа,старший байт которого в переменной RAMTOR, и необходимым для верной работы инструкции RETURN. Если во время ее выполнения стек GOSUB будет уже пуст, то это число сыграет роль его продолжения. Но поскольку оно больше чем 15872 (62*256). а строки бейсика не имеют такой нумерации, то это будет восприниматься как ошибка и сигнализировать сообщением RETURN без GOSUB. Сразу после этого байта (двигаясь вниз по памяти I находится стек GOSUB. В него заносятся номера программных строк, из которых были выполнены инструкции перехода к подпрограмме, чтобы интерпретатор знал адрес возврата по инструкции RETURN. Если интерпретатор не находится в подпрограмме, вызванной с помощью GOSUB, то этот стек просто не существует (точнее в нем не записано ни одного значения). Ниже находится МАШИННЫЙ СТЕК. используемый непосредственно микропроцессором. Оба этих стека заполняются в сторону уменьшения адресов памяти, особое значение имеет переменная ERRSP. Процедура, обрабатывающая ошибки бейсика (вызываемая командой процессора RST 8), помещает значение этой переменной в регистр SP. после чего выполняется RET? таким путем последний, записанный в стеке адрес (во время выполнения программы он обычно равен 4867). Под этим адресом находится процедура, выводящая сообщение об ошибке.

ОБЛАСТЬ ГРАФИКИ ПОЛЬЗОВАТЕЛЯ - это 168 байтов, зарезервированных для представления 21 графического символа, определенного пользователем. Адрес последней ячейки памяти (равный 65535 - если исправна вся память компьютера) запоминается в переменной P_RAMT. Если часть памяти повреждена, то эта переменная содержит адрес последней исправной ячейки.

3. СПОСОБ ЗАПИСИ ПРОГРАММ НА БЕЙСИКЕ

Практически каждая программа имеет хотя бы процедуру загрузчика, написанную на бейсике, если же она такой процедуры не имеет-то от вмешательства извне она не защищена При считывании простой бейсик-программы с ленты в течение 1-2 секунд идет пилот, затем кратковременно появляются тонкие мерцающие желто-фиолетовые полоски, свидетельствующие о том, что компьютер считывает в память информацию. Ее 17 байтов-это так называемый заголовок. Появляется надпись:
"BYTES: ", "PROGRAMM", "CHARAKTER ARRAY:" или "NUMBER ARRAY:", после небольшого перерыва начнется второй пилот (более короткий),а после него считывается собственно сама программа. Рассмотрим более подробно заголовки,так как в них содержатся основные сведения о считываемых программах.

Заголовок содержит 17 байтов- с 0-го по 16-й. Значения байтов приведены в таблице.

0	Тип блока:	0 - программа 1 - NUMBER ARRAY (числовой массив) 2 - CHARAKTER ARRAY (символьный) 3 - BYTES (данные)
1		
2		
3		
4	ИМЯ	
5		
6		
7		
8		
9		
10		
11	Длина	Количество байтов, которые необходимо будет считать с ленты
12		
13	Старт	BYTES: -адрес загрузки блока PROGRAMM: -номер строки пуска программы ARRAY: -14-й байт- имя переменной -13-й байт- не имеет значения
14		
15	Программа	PROGRAMM: -длина бейсика (без переменных) BYTES, ARRAY: - не используется
16		

Нулевой байт означает тип блока и равен:
0 - если программа на бейсике;
3 - если это блок машинного кода (записанный с помощью SAVE ". . . " CODE или SAVE "... " SKREEN$, которые означают то же, что и SAVE "..." CODE 16384,6912.
Если же этот заголовок предшествует набору, являющемуся массивом переменных бейсика (записанному с помощью SAVE ". .. " DATA . . .) , то байт содержит:
1 - для числовых массивов и
2 - для символьных.
Следующие 10 байтов -это имя считываемого блока или текст появляющийся после загрузки заголовка за надписью "PROGRAMM: ", "BYTES:" и т. д.
Байты 11 и 12 содержат двухбайтовое число (первый байт - младший), которое указывает длину блока, к которому относится заголовок.
В зависимости от считываемого блока байты с 13 по 16 интерпретируются по разному. В заголовках программ, написанных на бейсике, байты 13 и 14 содержат номер строки, с которой запускается программа - если она была записана с помощью SAVE "..." LINE NR. Если программа была записана без операции LINE и после считывания не запускается автоматически, то значение этого числа больше 32767. Одним из способов нейтрализации самозапускающихся программ на бейсике является замена этих двух байтов на число большее 32767.
Байты 15 и 16 содержат число, указывающее длину самой программы на бейсике (т.к.SAVE "..." или SAVE "..." LINE записывает программы со всеми переменными, т. е. содержимое памяти от байта, указанного системной переменной PROG, до байта, определяемого системной переменной E_LINE. Если от всей длины блока (байты 11 и 12) отнимем это число, то узнаем, сколько байтов в этом блоке занимают переменные бейсика. Все это сказано о заголовках программ на бейсике.

В заголовках блоков машинного кода ("BYTES") байты 15 и 16 не используются,а байты 13 и 14 составляют двухбайтовое число, указывающее, по какому адресу надо считать следующий за заголовком блок.

В заголовках массивов из этих 4-х байтов используется только 14-й байт, который представляет собой имя считываемого массива, ин записан так же как и имена всех неоеменных бейсика (в области dT VARS до E_LINE), т.е. три старших бита означают тип переменной, а пять младших битов-ее имя.

На листинге 1 приведена программа, позволяющая читать заголовки программ. Наберите программу,внимательно проверив количество пробелов в строках с инструкциями DATA, запустите ее RUN и подождите, пока на экране не появится информация об адресах начала и конца процедуры, а также о ее длине, которая должна составлять 284 байта. Если этого не призошло. проверьте, все ли строки введены без ошибок. Если все выполнено верно, то на ленту записана процедура

"ЧТЕН", позволяющая прочитать любой заголовок. После записи на ленту, из ОЗУ ее можно стереть.
Процедура "ЧТЕН" будет работать правильно независиио от того, по какому адресу она будет загружена. Можно считать ее: LOAD "CZYTACZ" CODE АДРЕС
а затем запустить с помощью RANDOMIZE USR АДРЕС

После запуска процедура считывает по адресу 23296 (буфер принтера) первый встреченный заголовок. Если из-за подключенных внешних устройств этот адрес нас не устраивает, можно изменить его, заменив в строке 200 листинга 1 число #005B шестнадцатиричным адресом, по которому мы хотим считать заголовок (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 4-х цифр контрольной суммы нужно поместить 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 необходимо просмотреть его дополнительно, то сделать это можно используя функцию PEEK.

Однако чтения заголовка мало для того чтобы прочитать блоки,записанные на ленте. Необходимо знать, как разместить эти блоки в памяти, не позволяя им при этом начать работу.

В случае блоков типа "BYTES" обычно достаточно загрузить их под принудит ельный адрес выше RAMTOR. Например:

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

Этот метод работает, если блок не очень длинный (не более 40K). Более длинные блоки могут не разместиться в памяти. Тогда нужно разделить их на несколько коротких частей. Позже мы посмотрим как это сделать. Также и в случае массивов - их загрузка не представляет трудностей -достаточно применить обычную в таких ситуациях инструкцию LOAD "" DATA ... . Сложнее выглядит считывание программ на бейсике. Они обычно записываются с помощью SAVE ". . . "LINE "...", а в самом начале - строка, с которой должны выполняться инструкции, закрывающие программу до останова. Простейшим решением является загрузка программы не с помощью LOAD. а с помощью MERGE "", но этот метод не всегда дает результат. Из этой ситуации есть два выхода: подменить заголовок программы, или использовать представленную ниже программа "LOAD/MERGE". Первый способ основан на замене заголовка записанной на ленте программы на такой же. но не вызывающий самозапуска программы. С этой целью можно использовать программу "COPY-COPY" -считать заголовок программы, нажать BREAK, затем инструкцией LET заменить ее параметр старта на число больше 32767 (т.е. выполнить например LET I=32768, если корректируемый таким образом заголовок был считан как первый набор). Измененный таким образом заголовок записыва-

ем где-нибудь на ленте. Убираем из памяти программу COPY-COPY и вводим LOAD. Считываем только что сделанный заголовок, и сразу после его окончания останавливаем ленту. Теперь вставляем в магнитофон кассету с программой - так, чтобы считать только текст программы без заголовка.

Другой способ значительно выгоднее первого. Введем в память (с клавиатуры или ленты) программу LOAD/MERGE, приведенную на листинге 2. После запуска она начинает ждать первую программу на бейсике, находящуюся на ленте, считывает ее совершенно так же как инструкция LOAD, но после загрузки не позволяет программе запуститься - выводит сообщение "0 OK" с информацией, с какой строки считанная программа должна была стартовать.

ЛИСТИНГ 1

```
   10 CLEAR 59999:LET POZ=60000
   20 LET ADR=POZ
   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 GOTO 130
   70 FOR N=1 TO LEN A$-5 STEP 2
   80 LET W=16*VAL A$(N)+A$(N+1)
   90 POKE ADR,W : LET ADR=ADR+1: LET S=S+W
   100 NEXT N
   110 IF VAL A$(N)<>S THEN PRINT"OSN.W STROK" : NR: STOP
   120 LET NR=NR+1:GOTO 60
   130 PRINT "VSE DANNYE VERNY";"NASHALO:";
POZ;"CONEC_:_";ADR-1;"DLINA_:_";ADR-POZ
   140 SAVE " CZYTACZ" CODE POZ,ADR-POZ
   150 REM
   200 DATA "21920009E5DD21005BDDE51111 1246"
   210 DATA "00AF37CD5605DDE130F23E02CD 1531"
   220 DATA "011611C009DD7E00CD0A0CDDE5 1265"
   230 DATA "D113010A00CD3C202A7B5CD1E5 1231"
   240 DATA "D5ED537B5C010900CD3C20DD46 1346"
   250 DATA "0CDD4EDBCD2B2DCDE32DE1DD7E 1872"
   260 DATA "00DD460EDP4E0DC5A7202BEB01 1292"
   270 DATA "1900CD3C20D5DD4610DD4E0FCD 1361"
   280 DATA "2B2DCDE32DD1C178E6C0206EC5 1848"
   290 DATA "011300CD3C20C1CD2B2DCDE32D 1280"
   300 DATA "185EFE03203601700009EB0111 836"
   310 DATA "0018E60D449C75676F9D9B2073 1281"
   320 DATA "516D65676F2070726F6772616D 1313"
   330 DATA "75200D4175747F737461727420 1161"
   340 DATA "2D206C696E69120D2EE1BB0181 1239"
   350 DATA "0009EB010900CD3C20C13E1FA0 997"
   360 DATA "F660D7DD7E003D28033E24D73E 1383"
   370 DATA "28D73E29D7E1227B5C3E0DD7C9 1538"
```

```
 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,
 54,1,225,205,29,7,42,66,92,34,69,92,207,255
```

ЛИСТИНГ 3

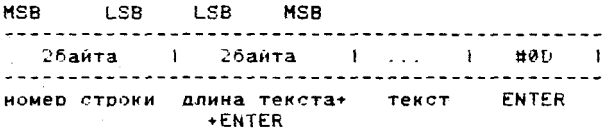
```
10;     LOAD/MERGE
20;
30      ORG     60000
40      LD      BC.34
50      RST     48
60      PUSH    DE
70      POP     IX
80      LD      (IY+58),1
90      LD      (IX+1),255
100     CALL    1821
110     LD      HL,(23618)
120     LD      (23621),HL
130     RST     8
140     DEFB    255
```

4. 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 вызывать EDIT ни для какой строки, видно подозрительную инструкцию RANDOMIZE USR 0, или просто ничего не видно,т.к. программа не позволяет листать себя. Если в программе, в которую мы вмешиваемся,заметно что-то необычное,то лучше просматривать ее другим способом, несколько отличающимся от обычного, не с помощью LIST, а непосредственно используя функцию PEEK.

Однако сначала нужно узнать, каким образом размещен в памяти текст программы на бейсике. Программа складывается из последовательных строк, и так и хранится в • памяти.

Отдельная строка программы выглядит так:

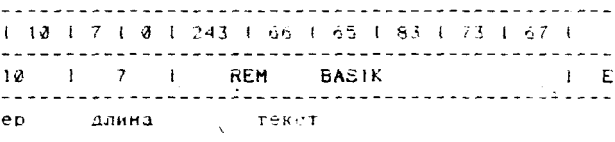


Она занимает не менее 5 (точнее 6. т.к. текст пустым быть не может) байтов. Два первых байта содержат ее номер, но он записан наоборот, по сравнению с традиционной записью двухбайтовых чисел, хранимых в памяти (MSB - старший байт, LSB -младший байт).

Следующие два 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 введем, например, такую строку:

10 REM BASIC

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



Параметр "длина строки" касается только ее текста, следовательно, хотя строка занимает в памяти 11 байтов, этот параметр указывает только на 7 байтов: 6 байтов текста и 1 баит "ENTER", заканчивающий строку. Теперь, наверное, понятен трюк со строкой, имеющей нулевой номер. Достаточно в первые два байта строки занести число 0 (с помощью POKE), чтобы эта строка стала нулевой строкои. Если мы хотим

изменить номер первой строки в программе (а интерфейсы ни к какой 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 останется в памяти там же, где и была. Если ввести:

```
 10 REM НОМЕР СТРОКИ 10
 20 REM НОМЕР СТРОКИ 20

 POKE 23735,0:POKE 23756,30
```

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

```
 30 REM НОМЕР СТРОКИ 10
 20 REM НОМЕР СТРОКИ 20
```

следовательно, чтобы начать разблокировать программу, в которой имеются нулевые строки или строки, упорядоченные по убыванию номеров, следует найти адрес каждой строки, и в их поле "номер строки" последовательно размещать,к примеру: 10,20,... . В памяти строки располагается одна за другой, следовательно с обнаружением начала строк особых трудностей не будет. 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 не действует, когда применяется другой способ зашиты - фальшивая длина строки. Он основам на том,что в поле "длина строки" вместо настоящего значения дается очень большое число- порядка 43-65 тысяч. Этот способ применяется очень часто, т.к. делает невозможным считывание программы с помощью MERGE (т.е. так, чтобы не было самозапуска) . Делается это потому, что MERGE загружает программу в область WORKSPACE, а затем интерпретатор анализирует всю считанную программу строка за строкой: последовательно проверяет номер каждой из них, а затем размещает ее в соответствующем месте области, предназначенной для текста программы на бейсике. Для этой строки необходимо там подготовить соответствующее количество свободных байтов "раздвигая" уже существующий текст программы. Если в поле "длина строжи" стоит очень большое число, то интерпретатор старается выделить именно столько байтов пространства в области текстов программы, что приводит к сообщению "OUT OF MEMORY" или просто зависанию системы. Чтобы прочесть такую программу, не вызывая ее самозапуска, следует, применить соответствующую программу, например такую, как представленная в предыдущем разделе программа "LOAD/MERGE".

Если программа защищена таким способом, то адреса очередных строк приходится искать вручную, или догадываться, где они находятся, помня, что каждая строка заканчивается знаком ENTER (но не каждое число 13 означает ENTER). Чтобы просмотреть программу на бейсике, введем такую строку:

```
FOR N=23627 TO PEEK 23767+256*PEEK 23628
PRINT;"    ";PEEK N; CHR$ PEEK N AND PEEK>31:
NEXT N
```

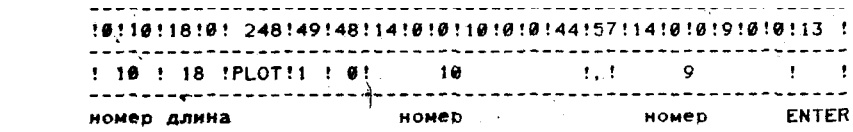
она последовательно высветит: адрес, содержимое байта с этим адресом, а также символ, имеющий этот код, если это только не управляющий символ (т.е. с кодом И...31). После смены нумерации строк и изменения их длин, следующим способом защиты программ являются управляющие символы, не позволяющие последовательно 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 и функции, а так же знаки <=, >= ,<>. - имеют однобайтовые коды из диапазона 165...255, (чтобы просмотреть, какие, можно ввести:

```
FOR N=165 TO 253 :PRINT N, CHR$ N: NEXT N
```

и ознакомиться с ними), 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 памяти без изменений. Иначе выглядит ситуация, когда в строке находятся другие инструкции, требующие числовых параметров (а обычно так и бывает). Введем, к примеру, строку:

```
10 PLOT 10,9
```

и посмотрим, каким образом она запишется в память (лучше вводя переменную выше строки H-23755 TO . . .) , выглядит она так, как показано на рисунке:



Как видно, текст был модифицирован - после последней цифры каждого числа, выступающего в тексте строки как параметр, интерпретатор сделал 6 байтов пространства и поместил там символ с кодом 14, а также 5 байтов, в которых записано значение этого числа, но способом, понятным интерпретатору. Это убыстряет, в определенной мере, выполнение программ на бейсике, т.к. оо время выполнения программы интерпретатор не должен каждый раз переводить числа из алфавитно- цифро-

вого представления (последовательность цифр) на 5-и байтовое представление, пригодное для вычислений, готовое значение выбирается из памяти п -ле управляющего символа CHRQ 14. Такая запись затру/н лет доступ к программам. Бо многих программах - загрузчиках (LOADER) присутствует такая строка:

```
   0 RANDOM IZE USR 0: REM . . .
```

На первый взгляд, эта программа после запуска должна затереть всю память, но на самом деле так не происходит. После более внимательного рассмотрения (с помощью PEEK - строка FOR N=23755 ...) оказывается после USR 0 и символа CHRU 14 совсем нет пяти нулей (именно так выглядит в пятибайтовой записи число 0). Если это целое число, то в пятибайтовой записи оно выглядит:

```
байт 1      - 0
байт 2      - 0 (для + I и 255 ( для - I
байт 3 и 4  - последовательно младший и старший
                ( или дополнение до 2 для - I
байт 5      - 0
```

к примеру 0,0,218,92,0 - это равнозначно числу 23770. Функция USR не осуществляется, тогда переход по адресу 0, а именно по 23770, а это как раз адрес байта, находящегося сразу после инструкции REM в нашем примере. Там обычно находится программа - загрузчик, написанная на машинном языке.

Следующим управляющим символом, часто применяемым для зашиты является CHRD 8 - "BACKSPACE" или пробел назад. Высвечивание этого символа вызывает сдвиг позиции ввода на одну позицию назад (влево) . Следовательно с его помощью можно закрывать некоторые позиции на листинге, печатая на их месте другой текст. Если, например, в памяти находятся знаки:

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

(< - означает СНГ.И 8), то инструкция LOAD"" и последующий текст закроют инструкцию LET A=USR 0:. Хотя на листинге видна только инструкция LOAD "":, но дальнейшая часть программы загружается не ею, а машинной программой, запускаемой инструкцией USR 0 - что. очевидно, не должно означать переход к адресу «. Такая зашита, например, применяется в загрузчике программы BETA BASIC 1..0 .

5 "ЗАЩИТНЫЕ" УПРАВЛЯЮЩИЕ СИМВОЛЫ

В этом разделе мы рассмотрим остальные управляющие символы. Три из них касается изменения места вывода, 6 - смены атрибутов.

Символ "УПРАВЛЯЮЩАЯ ЗАПЯТАЯ" -CHR$ 6 - "COMMA CONTROL" Этот символ действует так же как запятая, отделяющая тексты в инструкции PRINT, т.е. выводит столько пробелов (но не менее одного), чтобы оказаться в колонке 0 или 16:

```
PRINT "1", "2"
```

а также

```
PRINT "1"+CHR$ 6+"2"
```

имеют идентичное значение.

Символ "AT -УПРАВЛЯЮЩИЙ" - CHR$ 22 -AT CTRL Этот символ позволяет переносить позиции вывода в любое место экрана так же. как AT а 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 делать программы невидимыми < точнее листинги программ), введем, например:

```
10 RANDOMIZE USR 30000: REM  НИЧЕГО НЕ ВИДНО !
```

После инструкции REM введем три пробела, а после восклицательного знака две управляющие запятые. Их можно получить непосредственно с клавиатуры, нажимая последовательно клавиши: EXTENT (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 помошью POKE 23774,22: POKE 23775,0: POKE 23776,0 Попробуем теперь посмотреть программу. На экране не появится текст всей строки - начальная часть ее закрыта надписью, находящейся после инструкции REM и знака AT CHRL. Такие же трудности возникают, если эту строку перенести в зону редактирования (клавиша EDIT). Координаты, заданные в символе AT CHR$ должны находиться в поле экрана, т.е. номер строки не может быть больше 21, номер колонки - не больше 31. Задание больших значений в случае FRINT или LIST вызовет сообщение "OUT OF SCREEN" и отмену дальнейшего вывода. Если же 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 практический способ защиты от просмотра текста для каждой программы.

Символ "ГОРИЗОНТАЛЬНАЯ ТАБУЛЯЦИЯ" -CHR$ 23 - "TAB CTRL" После этого символа следуют два байта, определяющие номер колонки, в которую переносится позиция вывода. Они трактуются как одно двухбайтовое число (первый байт - младший). Поскольку 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 - также как и управляющая запятая и, следовательно, может быть использован для закрывания уже находящихся на экране текстов.

6. СИМВОЛЫ СМЕНЫ АТРИБУТОВ

Эту групгу управляющих символов составляют символы, меняющие атрибуты:

CHR$ 16 - INK CTRL
CHR$ 17 - PAPER CTRL
CHR$ 18 - FLASH CTRL
CHR$ 19 - WRIGHT CTRL
CHR$ 20 - INVERSE CTRL
CHR$ 21 - OVER CTRL

После каждого из этих символов обязателен один байт, уточняющий, о каком атрибуте идет речь, после символов INK и PAPER это могут быть числа 0...9, после FLASH и BRIGHT-0,1,8. после INVERSE и OVER - 0 и 1. Задание других значений вызывает сообщение "INVALID COLOR" и, естественно, прерывание просмотра программы.

Высвечивая программу, защищенную управляющими символами цветов, принимаем для 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 ***в.*** Кроме того, удаляем все мешающие знаки BACK SPACE (CHR$ 8) путем их замены пробелами (CHR$ 32). Так же ликвидируем знаки AT CTRL - заменяем с помошью POKE три байта символов на пробелы. После такой корректуры программу можно уже листать без всяких сложностей.

Если нужно прочитать программу загрузчика, то его не обязательно очищать - важно узнать, что эта программа делает, каким образом загружается в память и запускает следующие блоки, а не стараться, чтобы она работала "ровно" и была написана "логично". Это тем более важно, что пока не узнаешь точно программу. лучше не делать в ней никаких изменении - одна ловушка может проверяться другой, поэтому наилучший способ изучения программы -это анализ ее работы шаг за шагом, считывая последовательные байты памяти:

```
   0 BORDER 0: PAPER 0: INK 0: CLS 0: 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;:
   RANDOMIZE USR 24064
```

Необходимо помнить о правильной интерпретации очередных байтов:
сначала два байта номера строки:
затем два байта длины программы,которая, кстати, может быть фальшивой;
затем текст: инструкция бейсика, потом ее параметры. за каждым числом записываются CHR$ 14 и пять байтов, содержащих значение этого числа;
за параметрами - двоеточие и следующая инструкция или ENTER и новая строка программы.

Это было бы все, если бы речь шла об управляющих символах. Но есть еще одна вещь, которую требуется знать,чтобы не иметь неприятностей с чтением бейсика. Речь идет об инструкции DEF FN. Введем и внимательно рассмотрим строку:

```
10 DEF FN A(A,B$, C) =A + C
```

На первый взгляд она должна занять в памяти 19 байтов (номер строки, ее длина, ENTER,а также 14 введенных символов), но это не так. Интерпретатор после каждого параметра функции поместил знак CHR$ 14 и дорезервировал зачем-то следом еще пять байтов. Введем:

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

и снова просмотрим содержимое памяти с адреса 23755. После первого параметра в определении функции Находятся CHR$ 14, но после него последовательно расположились: 0,0,1,0,0, - что в памяти байтовой записи обозначает 1. Также после третьего параметра функции находится CHR$ 14 и байты, содержащие число 2. После параметра B$ также находится значение использованного параметра: CHRQ 14 и пять байтов, которые последовательно содержат: первый - для нас не имеет значения, второй и третий - содержат адрес, по которому находится цепочка символов "125" (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 три знака.

Необходимо помнить об этом, читая бейсик с помощью PEEK, 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 (пример - BETA BASIC 1.0.).

завершение раздела немного о программах-загрузчиках. Их задачей является считывание и запись всех блоков, составлявших программу. Обычно они это делают способом.

максимально затрудняющим понимание их работы - так, чтобы запуск программы другим способом, а не через загрузчик (или вмешательство а программу) был невозможен. Посмотрим на загрузчики, применяемые в большинстве программ фирмы ULTINAME (например ATIC ATAC, KUINGHTLORE, PENTAGRAM, NIGHT SHADE и т.д.). Выглядят они примерно так:

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

После такого загрузчика на ленте находятся пять следующих блоков: экран, закодированный блок программы, а за ним три коротеньких блока, защищающих программу: однобайтовый (код инструкции JP(HL)), из нескольких байтов (это процедура, которая декодирует всю программу), и последний двухбайтовый, загружаемый по адресу 23627, или в переменную FRAMES. Значение этой переменной увеличивается на 1 через каждые 1/50 секунды. В машинной программе, запущенной с помощью RANDOMIZE 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. Вмешательство в программу такого типа весьма просто. Достаточно загрузить все блоки за исключением последнего, а после просмотра программы или проведения в ней определенных изменений (например,вписания POKE) достаточно лишь ввести:

```
LOAD "" CODE: RANDOMIZE USR 24064
```

(но обязательно в одной строке, разделяя инструкции двоеточием), чтобы запустить программу.

Отдельной работой является декодирование программы, или запуск процедур, декодирующих так, чтобы она вернулась в бейсик. У читателя, знающего ассемблер, это не должно вызвать затруднений. Однако к вопросу распространенности этого типа защиты, особенно в пользовательских программах (например ART STUDIO или THE LAST WORD) . мы еще вернемся.

7. ЗАЩИТА ЗАГРУЗЧИКОВ

Все игры имеют хорошо защищенную программу, написанную на бейсике. Это важнейший с точки зрения действенности зашиты элемент всей программы, т.к. с бейсика начинается считывание всей программы. Если бейсиковский загрузчик защищен слабо, то легко прочитать и программу , как это было показано на примере загрузчиков фирмы ULTINAME. Одним из способов снятия зашиты загрузчиков является считывание их с помощью программ "LOAD-MERGE", описанных выше. Однако иногда бывает лучше поместить этст загрузчик не в память, предназначенную для бейсика, а выше RAMTOR, чтобы можно было рассматривать его не опасаясь возможности случайных изменений в нем. Для этого есть очень действенный

метод-считать программу на бейсике как блок машинного кода под удобный для нас адрес. Для этого надо знать длину программы,которую необходимо считать (можно воспользоваться процедурой CZYTACZ, описанной выше). Кроме того нужно иметь немного свободного места на магнитной ленте. Этот способ основан на обмане инструкции LOAD путем подмены заголовков. На ленте нужно записать заголовок блока кода с помощью SAVE "BAS" CODE 30000,750 - если известно что длина программы, например, 790 байт. Если длина неизвестна - задается значительно большая длина блска по сравнению с возможной реально. На ленте записывается только сам заголовок и запись прерывается. После этого нажимаем клавишу BREAK, установим 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 нажать в магнитофоне СТОП, заменить кассету и нажать ПУСК (все это время компьютер ждет блок данных). Теперь считыеается программа на бейсике, но под адрес 30000 - выше RAMTOR. Если в заголовке мы задали большую длину программы,чем требовалось, то считывание закончится сообщением "TARE LOADING ERROR",но это не имеет значения.Теперь можно любым способом смотреть считанную программу, хотя бы набирая для этого программу на бейсике.

Кроме этого метода есть и другой, но для пользования им необходимо знание ассемблера.

Нередко, особенно в новейших программах, встречаются блоки программ, записанные и считываемые в компьютер без заголовка. Эта довольно оригинальная мера защиты часто отпугивает начинающих, хотя ничего сложного в ней нет. Вся хитрость основана на хранящихся в ПЗУ процедурах, используемых с помощью инструкций LOAD, SAVE, VERIFY,MERGE. Под адресом 0556 (1366) находится процедура LOAD-BYTES, считывающая с магнитофона блок данных и следующую за ним информацию. При этом безразлично, будет ли это заголовок или требуемый блок данных, которые нужно поместить где-то в памяти.

Повторим, что каждая защищенная программа начинается с загрузчика, написанного на бейсике. Программа, применяющая загрузку без заголовков (с помощью процедуры 1366 или аналогичной), должна быть написана в машинном коде- как и всякая программа, обслуживающая магнитофон. Обычно такая программа помещается в одной из строк бейсика, например после инструкции REM, или в области переменных бейсика. После считывания, загрузчик на бейсике запустится и выполнит инструкцию RANDOMIZE USR.... инициализируя тем самым исполнение машинной программы. Процедура LOAD-BYTES требует задания входных параметров. Они задаются в соответствующих

регистрах микропроцессора. Так в регистре IX задается адрес, по которому хотим записать блок данных, в паре DE -длина этого блока. В буфер помешаем 0 -если хотим записать заголовок, и 255 -если это блок данных. Указатель переноса CARRY устанавливаем в 1, иначе процедура 1366 вместо LOAD выполнит VERIFY.
Ниже приведен пример загрузки с ленты без заголовка.

```
LD      IX,16384;       адрес считывания
LD      DE,6912;        длина блока
LD      A,255;          блок данных
SCF;                    установка CARRY
CALL    1366;           вызов LOAD-BYTES
RET;                    выход из подпрограммы
```

Процедура 1366 в случае ошибки считывания не выводит сообщение "TAPE LOADING ERROR", но существует еще одна процедура загрузки, которая это делает. Она находится под адресом 2050 и выглядит так:

```
2050    CALL    1366;   считывание блока данных
2053    RET     C;      возврат - если не было ошибки
2054    RST     8;      иначе RST 8 с сообщением
2055    DEFB    26:     "TAPE LOADING ERROR"
```

После возврата из процедуры 1366 указатель переноса содержит информацию о правильности считывания блока. Если он удален, то это означает что произошла ошибка. Некоторые загрузчики используют именно процедуру 2050 а не 1366.

Иногда загрузчики не используют ни ту ни другую процедуру, а заменяют их собственными,правда обычно похожими на процедуру 1366 или являющимися ее переделкой, благодаря которой, например, блоки данных загружаются в нижнюю часть памяти. с больших адресов к меньшим, или загрузка идет с другой скоростью. Такую программу следует анализировать с помощью дизассемблера, сравнивая ее фрагменты с тем. что находится в ПЗУ.

Поясним порядок использования процедуры из ПЗУ для считывания беисика по любому адресу, а не в предназначенную для него область. С помощью CZITACZ прочитаем заголовок программы, которую мы хотим прочитать, и запомним ее длину вместе с переменными. Затем ВЕедем программу на ассемблере, которая считает бейсик под установленный адрес (выше RAMTOR):

```
LD   IX,   АДРЕС
LD   DE,   ДЛИНА
LD     A,255
SCF
JP     2050
```

Так же как й при подмене заголовка, если неизвестна длина прграмны - задаем ее завышенное значение, но при этом чтение завершится сообщением TAPE LOADING ERROR. Каждый раз считывать ассемблер, чтобы ввести программу, приведенную выше, достаточно хлопотно, поэтому лучше создать ее на бейсике с помощью POKE:

```
 10 INPUT "АДРЕС ЧТЕНИЯ BASIK?";A
 20 RANDOMIZE A: CLEAR A-1
 30 LET A=PEEK 23670: LET B=PEEK 23671
 40 LET ADR=256*B+A
 50 INPUT "ДЛИНА BASIK?";C
 60 RANDOMIZE C: LET C=PEEK 23670
 70 LET D=PEEK 23671
 80 FOR N=ADR TO ADR+11
 90 READ X: POKE N,X
100 NEXT N
110 DATA 221,33,A,B,17,C,D,62,255,195,2,8
120 RANDOMIZE USR ADR
```

Установить ленту с обрабатываемой программой за заголовком программы. Запускаем программу, приведенную выше, вводим данные и включаем магнитофон. Результат аналогичен Тому который получали при подмене заголовков, но при этом не создается беспорядка на кассетах.

В завершение отметим еще одну процедуру, размещенную в ПЗУ по адресу 1218. Это процедура SAVE-BYTES - обратная LOAD-BYTES, т.е. записывающая на ленту блок с заданными параметрами: перед ее выполнением в регистр IX записываем адрес, с которого начинается здесь запись, DE содержит длину записываемого блока. В буфер запишем 0 - если это должен быть заголовок, или 255 - если это блок программы. Состояние указателя CARRY значения не имеет.

8. ИСПОЛЬЗОВАНИЕ СИСТЕМНЫХ ПРОЦЕДУР

Ранее были представлены процедуры из ПЗУ: SAVE7BYTES и LOAD-BYTES. Рассмотрим как использовать эти процедуры для защиты программ. Рассмотрим блоки машинного кода, запускаемые нестандартно.
Первый способ- это прикрытие загрузчика блоком, который им загружается. Такая защита применена, к примеру, в игре ПАРАДИЗ-СИТИ. Проследим способ чтения этой программы, так, чтобы она не стартовала автоматически. Начнем с бейсика. Практически наиболее важной инструкцией является RANDOMIZE USR. Лучше всего восстановить эту прцедуру с адреса запуска RANDOMIZE USR (PEEK 23647+256*PEEK 23628), т.е. в нашем случае с адреса 24130. Подобные процедуры используют известную нам процедуру 1366, считывающую блоки без заголовка.

Также и в этом случае, но перед ее вызовом с помощью команды LDIR процедура загрузки переносит сама себя в конец памяти по адресу 63116 и переходит туда по команде JP:

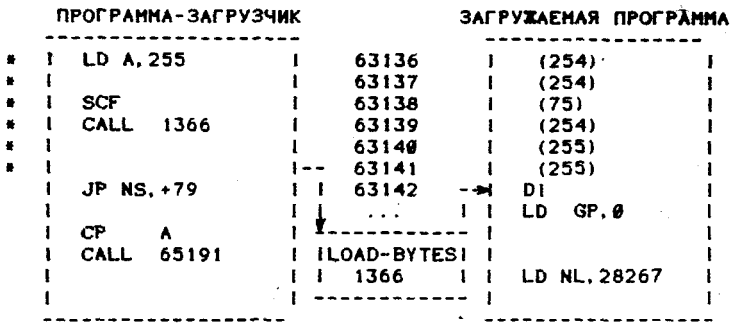
```
24130           DI;             запрет прерывания
24131   LD;      SP,0;          LD SP, 65536
24134   LD       HL,(2367);     в HL адрес на 28
24137   LD       DE,28;         больше чем значение
24140   ADD      HL,DE;         переменной VARS
24141   LD       DE,63116:      в DE адрес, а в
24144   LD       BS,196;        BC длина блока
24147   LDIR;
24149   JP       63116;         продолжение выполнения
                                программы с другого
24152   LD       IX, 16384;     адреса
24156     LD     DE,69212;
```

потом считывается картинка, а затем главный блок данных:

```
63116   LD       IX,16384;      подготовка загрузки
63120   LD       DE,6912;       картинки на экран
63123   LD       A,255;         с помощью процедуры
63125   SCF;                    LOAD-BYTES из ПЗУ
63126   CALL     1366;
63129   LD       IX,26490;      параметры главного
63133   LD       DE,38582;      блока программы,
63136   LD       A,255;         который считываясь,
63138   SCF;                    затирает эту процедуру
63139   CALL     1366;          считывания блока

63142   JP       NZ,+79         после возврата из
63144   CP         A;           процедуры 1366 здесь
63146   CALL     65191          уже находится другая
63149   JP       NC,-2          программа
```

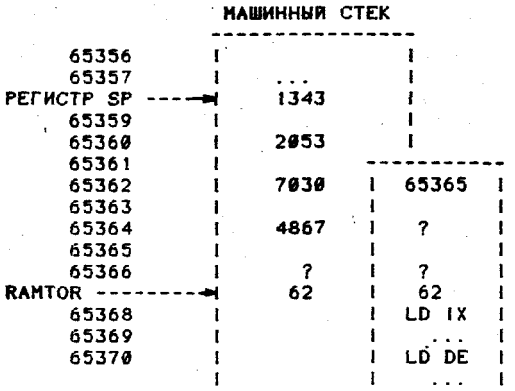
Способ запуска считанной программы требует пояснения. Как известно, каждая инструкция CALL заносит в машинный стек адрес, с которого начинает работать программа после выхода из подпрограммы. В этом загрузчике после выполнения второй инструкции CALL 1366 в стек заносится адрес команды за CALL. т.е. 63142,и процедура загрузки затирает сама себя, так как считывает байты с магнитофона в ту область памяти, где она была размешена. Важное значение имеет способ запуска считанной программы. Процедура 1366 заканчивается инструкцией RET. означающей переход по адресу , записанному в стек, или, в нашем случае, по адресу 63142. В процессе считывания программы процедура, которая находилась там, была замешена считанной программой, но микропроцессор этого не замечает и возвращается по адресу, с которого выполнена CALL 1366, не обращая внимания на то, что там находится уже другая программа. Схематично это показано на рисунке:



Слева расписано содержимое памяти до, а справа- после считывания программы. Команды,отмеченные " * " ложатся на выполняемую программу.

Как распознать защиту такого рода и как ее обойти? Начнем с бейсика. Считываем загрузчик (на ассемблере) и определяем адрес окончания считанных блоков (добавляя к адресу начала - регистр IX длину блока -регистр DE). Если какой-либо из блоков накрывает процедуру загрузки, то это и означает, что программа считывается и загружается именно таким способом. Дальше достаточно, опираясь на данные об адресе и длине блока, написать короткую процедуру, загружающую нужный нам блок кода, или подготовить соответствующий заголовок, а затем с помощью CLEAR ADR установить соответствующим образом машинный стек (так, чтобы машинная программа не уничтожила стек), и, после этого, считать программу. После выполнения в ней необходимых изменений, ее можно записать на ленту, но в том же виде, в каком был записан оригинал (длина блока должна совпадать прежде всего !) . Если этот блок был без заголовка (как в нашем случае) , то записываем его обычным SAVE ". . . " CODE ... , но опуская заголовок, т.е. магнитофон включаем только в перерыве между заголовком и блоком кода. Можно также попробовать запустить считанный блок переходом на требуемый адрес командой RANDOMIZE USR ... , но это не всегда получается. В игре "ТРИ НЕДЕЛИ В ПАРАДИЗ СИТИ" этим адресом будет 63142, и, можно убедиться - этот метод не срабатывает.

Другим интересным способом запуска блоков машинного кода является считывание программы в область машинного стека. Таким способом можно запускать блоки машинного кода, загружая их просто через LOAD " " CODE. Схематично этот метод показан на рисунке:



Указатель стека (регистр SP) принимает показанное на рисунке состояние в процессе выполнения процедуры 1366 (вызванное из бейсика через LOAD""CODE). Способ запуска программы в целом прост. Адрес считывания блока рассчитан так, что блок считывается на машинный стек именно с того места, в котором находится адрес возврата из инструкции LOAD""CODE (он при этом равен значению системной переменной ERRSP-2) или прямо из процедуры LOAD-BYTES (равен ERRSP-6). Тогда два первых байта программы содержат 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 микропроцессор считывает содержимое стека и переходит по прочитанному адресу, который только что появился в памяти. По этому адресу в программе находится начало процедуры загрузки ее последующих блоков - что и показано на рисунке.

Для того,чтобы обойти такую защиту, достаточно заменить RAMTOR на соответственно низкое значение, затем прочитать блок кода, который из-за произведенной замены не запустится. - Ситуация может осложниться, если блок окажется очень длинным (что случается очень редко,но случается) . В этом случае мы должны поступить с ним так же, как и с любым другим длинным блоком, но помнить, с какого адреса он запускался.

Теперь рассмотрим вопрос разделения на части блоков длиной более 42K. Вмешательство в блоки такого типа основано на разделении их на фрагменты так. чтобы в памяти еще оставалось место для MONS-A или другого дизассемблера, исправления этих фрагментов, а затем сборки их в целый блок либо написания новой процедуры загрузки. Обычно достаточно разделить блок на две части. Вначале необходимо из процедуры загрузки, или, если ее нет. то из заголовка этого блока

получить длину блока и адрес его загрузки. Чтобы получить первую часть блока используем процедуру 1366 но с другими параметрами- не с теми, которых требовал разделенный на части блок,просто задаем адрес, по которому хотим разместить этот блок (выше RAMTOR), а также длину - примерно 16К, несмотря на то, что блок этот значительно длиннее. Считаем теперь этот блок через CALL 1366 или CALL 2050, но во втором случае сообщение "TAPE LOADING ERROR", которое появится, не даст нам никакой информации о верности считывания, т. к. мы загружаем часть блока без контрольного байта, находящегося в конце блока. Считанную таким образом первую часть блока записываем на ленту и сразу же приступаем ко второй части. Ее считывание сложнее ввиду ограничений по памяти, но также возможно. Вспомним, что в нашем компьютере 16К ПЗУ, запись в которые невозможна. Например, вызовем процедуру 1366 с адресом чтения равным 0, и начальные 16К будут просто потеряны, а в память ОЗУ считаются только следующие 32К или меньше (т.е. оставшаяся свыше 16К часть блока). Считываемый блок разместится в ОЗУ с адреса 16384, заходя на системные переменные длины блока и оставляя без изменения лишь 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. Нужно 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 1366, благодаря чему после считывания блока не будет выполнена процедура 1343. Именно она еще и разблокирует прерывания. С помощью CLEAR 64999 переносим машинный стек, с адреса 65500 помещаем процедуру загрузки:

```
ORG       65000;
LD        IX,0;          адрес считывания
LD        DE,DL;         длина блока
LD        A,255;         подготовка к считыванию
SCF;                      блока
INC       D;             таким способом
EX        AF,AF;         заменяем начало
DEC       D;             процедуры 1366
DI;                      а затем входим
LD        A,15;          в ее середину:
OUT       (254),A;
CALL      1378;
LD        A,0;           черная рамка
```

```
          JP          C,OK;          будет означать
          LD          A,7;           верное считывание
OK        OUT         (254),A;       белая - ошибочное
CZEKAJ    LD          A,191;         ожидаем нажатия
          IN          A,(254) ;      "ENTER"
          RRA;
          JP          C,CZEKAJ;
          LD          IX,0;          запись считанного
          LD          DE,DL-16384;   блока на
          LD          A,255;         ленту
          CALL        1218;          а также
          LD          HL,64999;      инициализация
          JP          4633;          системы бейсика .
```

Вместо того, чтобы считывать ассемблер и вводить эту программу, можно запустить на ейсике программу, представленную ниже:

```
10 CLEAR 64999
20  INPUT "DLINA BLOKA?"; DL:RANDOMIZE DL
   :LET X=PEEK 23670: LET Y=PEEK 23071: LET S=0
30 FOR N=65000 TO 65054: 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 О KEY,  ВКЛЮЧИ МАГНИТОФОН"
60 RANDOMIZE USR 65500
70 DATA 221,33,0,0,17,X,Y,62,2,55,55,20,8,21,243,62,
        15,211,254
80  DATA  205,98,5,62,0,56,2,62,7,211,254,62,191,219,
        254,31,56
90 DATA 249,221,33,0,0,17,X,Y-64,62,255,205,194,4,33,
        231,253,202,25,18
```

Рассмотрим, как разделить блок. Установим ленту на блоке кода, который нужно разделить. Если он имел заголовок, то его опускаем. Запускаем процедуру и включаем магнитофон, в определенный момент времени увидим, что программа считывается на экран - так и должно быть. После загрузки блока цвет рамки говорит о правильности считывания: черная рамка- все в порядке, белая - была ошибка. Вставим в магнитофон другую кассету, включаем запись и нажимаем "ENTER" Программа запишется на ленту, 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.

Чтобы запустить измененную программу придется немного

потрудиться и собрать разделенную программу или написать для нее новую процедуру загрузки. Если программа заполняла полностью 48К памяти ОЗУ, возможен только второй метод.

Если надо соединить блоки - достаточно написать процедуру, похожую на разделяющую, но такую, которая считывает первый блок под адрес 16384, второй - сразу за ним, а затем запишет их вместе как один блок.

6 этом разделе был описан способ запуска блоков машинного кода путем считывания а область машинного стека. Для запуска такого блока достаточно ввести инструкцию LOAD""CODE, которая и запустит его.Чтобы убедиться в этом на практике, запустим программу, приведенную ниже:

```
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 "ОШИБКА В СТРОКЕ DATA"
  :стоп
40 SAVE "BEEP" CODE 65362,34
50 DATA 88,255,3,19,0,62,221,2,
        29,6,8,197,17,30,0,33,0,8,43,124,18
60 DATA 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 освободим память с помощью RANDOHIZE USR 0 или RESET (по возможности, нужно переставить RAMTOR в нормальное значение с помощью CLEAR 65367) и прочтем его, введя LOAD""CODE. Задача этого блока - проинформировать, что он запустился - что и делается несколькими звуковыми сигналами, которые появятся сразу после считывания программы, как только с рамки исчезнут гранатово - желтые полосы.

9. ДЕКОДИРОВАНИЕ ЗАКОДИРОВАННЫХ БЛОКОВ ТИПА "BYTES"

Кодирование программы - это род достаточно простого шифра, делающего невозможным правильную работу программы. Для ее запуска служит специальная декодирующая процедура. которая находится в той же программе, и, что самое важное- не закодирована. Кодирование просто затрудняет доступ к тексту программы,после того как все предшествующие зашиты устранены и программа считана без самозапуска. В этом случае в памяти лежит "полуфабрикат" программы, который только после обработки декодирующей процедурой становится программой. Кодирование может быть основано 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 76000,

30672) частично закодирован. По адресу 26000 находится инструкция JP 26024, которая осуществляет переход к декодирующей процедуре:

```
26024   21C154  LD HL,26049:     запись адреса
26027   ES      PUSH HL;         26049 в стек
26028   21C165  LD HL,26049;     адрес начала
26031   11476C  LD DE,26719;     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);          XOR и запись .
26041   23      INC HL;          следующий адрес
26042   B7      OR A;            проверка признака
26043   ED52    SBC HL.DE:       конца, возврат в
26045   19      ADD HL.DE:       цикл или выход
26046   20F2    JP NZ,26034;     по адресу, записан-
26048   C9      RET;             ному е 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 декодированного блока, в DE - 27719. как адрес последнего декодированного блока. Затем в цикле декодируются последовательно байты-инструкция SUB 34, RLCA, и XOR #CC 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 записсанным в стек является не адрес возврата в бейсик, а записанный в начале с помощью PUSH HL адрес 26049 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 двух разных входных данных результаты тоже различны. RLCA заменяет значения битов 7,6, 3 и 2 на противоположные. Другими инструкциями, имеющими те же самые свойства яеляются. например. ADD, ING, DEC, RRCA, NEG, CPL но не OR или AND. Как раскодировать этот блок? Проще всего - вводя в бейсике:

```
POKE  26027,0  (код инструкции NOR)
```

Тем самым ликвидируя инструкцию PUSH (RET в конце программы перейдет в бейсик). и выполнить RANDOMIZE USR 26000. Однако помнить о том. что декодирующая программа может проверяться другим фрагментом программы. Вот дальнейшая часть программы в "ART STUDIO" :

```
26283  67        LD H, A;          в A уже находится
26264  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..27027
26301  B7        OR A;             и если оно другое-
26302  11C165    LD DE26049;       то стирание
26302  ED52      SBC 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        POR 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 записан адрес 0).
Что делать в этом случае? Ножно 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 так:

```
26003    21C165  LD      HL.26;      адрес начала
26006    11476C  LD      DE,27719;   адрес конца
26009    7E      LD      A, (HL) ;   выборка байта из
26010    EECC    XOR     #CC;        памяти, кодирование
26012    OF      RRCA;               его с помощью XOR
26013    C622    ADD     34;         RRCA, ADD и его
26015    77      LD      (HL).A:     запись
26016    23      ING     HL;         следующий адрес
26017    B7      OR      A;          проверка окончания
26018    ED52    SBC     HL,DE;      переход в цикл или
26020    19      ADD     HL, DE;     возврат в
26021    20F2    JR      NZ,26009;           бейсик
```

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

При защите программ применяются также малоизвестные и не опубликованные в фирменных каталогах команды микропроцессора Z80. Благодаря их применению программа становится малочитаемой, да и просмотр ее дизассемблером затруднен.

Наиболее часто встречаемыми не опубликованными инструкциями являются команды, оперирующие на половинках индексных регистров IX и IY в группе команд, которым не предшествует никакой иной префикс (т.е. SBH 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 HX, через IX - младшая. Аналогично HY и IY. Вот пример:

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 C,H	DD64	LD HX,HX	FD64	LD HY,HY
2601	LD H,1	DD2601	LD HX,1	FD2601	LD HY,1
B5	OR L	DDB5	OR IX	FDB5	OR IY

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

Префикс FDH или DDN относится ко всем регистрам H,L или HL, присутствующим в команде, следовательно, в одной инструкции невозможно использование ячейки адресованной как (HL), регистра HL. H или L одновременно с HX, HY, IX, IY (в дальнейшем ограничимся регистром IX, но все это относит:" и к регистру IY I. например:

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

Несколько иначе представляется ротация ячейки, адресуемой индексным регистрам, т.е. инструкцией начинающейся кодом DDCB. Инструкция типа RR (IX+***) и им подобные подробно описаны во всех доступных материалах о микропроцессорах Z80, но мало кто знает об инструкциях типа RR (IX + * * *),% и им подобных, где % обозначает любой регистр микропроцессора, они основываются на префиксировании кодом DDH или FDH 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 затем пересылки результата как в ячейку (IX+*), так и в соответствующий внутренний регистр микропроцессора. Например:

```
CB13                    RL E
DDCB0113                RL (IX + 1) ,E
DDSB0116                RL  (IX+1)
DDSE01                  LD E,  ( IX+1)
```

В конце рассмотрения инструкции этого типа следует вспомнить что команд EX DE, IX или EX DE. IY нет. Префиксирование команды EX DE, HL не дает никаких результатов. Также и 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, H RRCA и т.д.).

Еще одной интересной 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 сдвигам:

```
CC37                    SLI          A
CB3б                    SLI          (HL)
DDCB«*36                SLI          (IX+*)
DDCB«*57                SLI          (IX+*),A
```

Временами некоторые проблемы вызывают построение Флажкового регистра, особенно тогда, когда он используется достаточно нетипично. Например:

```
PUSH                    AF
POP                     BC
RL                      C
JP                      NC,...
```

Его вид представлен на рисунке

7	6	5	4	3	2	1	0
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 восьмибитовой арифметической или логической операции IN%,(C) и IN F,(C). Например:

```
XOR         A;          запись значения 0
ADD         A, 15;      результат - 00001111
указатели будут : F5=0 : F3=1
```

Еще одна возможность построения зашиты основана на использовании регистра обновления памяти P, а точнее того, что когда после очередного машинного цикла микропроцессора инкрементируется значение этого регистра, то его старший бит остается неизменным, следовательно, может быть использован для хранения любой, естественно однобитовой информации. Важно так же 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.

Б конце несколько коротких, но часто применяемых мер защиты. Их ликвидация основывается обычно на действиях обратных защитным. Например, переменная (DF SZ (23659) определяет число строк в нижней части экрана, требуемое для вывода сообщения об ошибке или для ввода данных. Во время работы программы это значение равно двум, но достаточно занести туда 0, чтобы программа зависла при попытке прерывания (т.к. выводится сообщение, для которого нет места). Если в программе находится инструкция INPUT или CLS, то она имеет подобный результат.

Распостраненным способом защиты является также изменение значений переменной BORDER (23624), которая определяет цвет 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, что и у рамки, следовательно, при остановке программы, можно подумать, что она зависла (Т.к. не видно сообщения). Разблокировать программу можно вписав BORDER 0 (или другой цвет).

Еше одним способом защиты может быть изменение переменной ERRSP (23613-23614), или дна машинного стека.(эта переменная указывает дно стека), где обычно находится адрес процедуры обрабатывающей ошибки языка BASIC (вызываемой с помощью RST 8). Уменьшение значения ERRSP на 2 вызывает защиту программы от "BREAK" и любой другой ошибки-программа заново запускается с того места в котором произошла ошибка. Смена содержимого дна машинного стека или другая замена значений ERRSP может вызвать зависание или даже рестарт компьютера.

Известным способом защиты является также занесение значения больше 9999 в ячейки памяти, обозначающие номер строки бейснка (первый байт - старший). Если он размешается в границах 10000 - 16383. то на листинге это выглядит, например, 0000 (вместо 10000) и строки невозможно скорректировать (EDIT). если же превышает 16384 - дальнейшая часть

программы считается несуществующей. Можно также встретить защиту основывающуюся на занесении минимальных значений (т.е.1) в переменные REPDEL (23561) и REPPER (23562), что затрудняет работу с компьютером, но для ее ликвидации требуется всего лишь быстрая реакция компьютера (запишите с помощью POKE нормальные значения : 35 и 5 - соответственно).