ZX - РЕВЮ
N1,N2 1993

"ИНФОРКОМ" 123019, Москва, Г-19, а/я 16

СПЕКТРУМ В ШКОЛЕ

Для тех, кто использует компьютеры семейства "ZX-Spectrum" в школьных учреждениях, мы подготовили сегодня программу "МОЗАИКА". Интересным, на наш взгляд, в ней является то, что это программа двойного назначения. Ее можно применять на уроках информатики как в старших классах, так и в младших.

Более опытные ученики смогут набрать текст этой программы и отладить ее, а для младших школьников она явится замечательным первым шагом в компьютеризацию, поскольку поможет легко наладить процесс общения с компьютером и сделать его радостным.

У нас есть положительный опыт работы с этой программой шестилетних детей, которые без труда освоили всю необходимую технику управления.

Не лишним также будет отметить и тот факт, что применима эта программа не только в школе, но и конечно дома. В этом случае она доставит много приятных минут и детям и их родителям. Не пожалейте несколько утренних часов. Ваш сын или дочь на всю жизнь запомнят этот воскресный день.

Дополнительный интерес в этой программе представляет и то, что при ее наборе Вам предстоит ознакомиться со многими профессиональными приемами, например с подгрузкой своего знакогенератора, с организацией перехода по ошибкам ON ERROR GO TO, с работой шестнадцатиричного загрузчика, с организацией в программе нулевой строки, с заполнением этой строки блоком кодов REM FILL, с адаптацией программы под БЕТА-ДИСК-интерфейс и т.д. и т.п. Все эти вопросы здесь последовательно разобраны, но Вам могут (если Вы пожелаете) пригодиться ранее выпущенные нами сборники "ZX-РЕВЮ" за прошлые годы.

"ИНФОРКОМ" предоставляет Вам возможность заказать и получить по почте сборники "ZX-РЕВЮ-91" и "ZX-РЕВЮ-92", выполненные в виде аккуратных книжек стандартного формата. Наш адрес Вам известен.

Те, кто не хочет связываться с почтой, смогут приобрести их лично на нашем корпункте по адресу: г. Москва, ул. Новый Арбат (бывший просп. Калинина), д.2, о/с Г-19. На первом этаже Вы найдете нас по вывеске "ZX-РЕВЮ".

ПРОГРАММА "МОЗАИКА"

В магазинах игрушек продается игра под названием "МОЗАИКА". Кто в детстве не играл в нее? В панель с отверстиями вставляются разноцветные фишки, которые образуют рисунок. Сегодня вниманию читателей предлагается компьютерный вариант этой игры. Программа может использоваться не только в младших классах школы, но даже в дошкольных учреждениях.

По существу это простейший в управлении графический редактор. Освоить профессиональный графический редактор, такой, как, например "ARTIST" или "ARTSTUDIO", ребенок не в состоянии, а программа "МОЗАИКА" поможет ему приобщиться к "компьютерной графике". В то же время, эта программа приучит ребенка пользоваться функциональными клавишами переключения цвета чернил, бумаги, режима работы "графика - текст" и т.п., то есть будет пройден "курс подготовки" перед освоением в будущем более сложных программ.

Для того, чтобы в деталях понять многие моменты, используемые в этой программе, следует обратиться к циклу ряда статей, опубликованных под рубрикой "Профессиональный подход" в ZX-РЕВЮ в 1992 году (см. стр. 113-116, 193-202, 241-248). Однако, если у Вас нет возможности ознакомиться с этими материалами, или нет желания вдаваться в подробности работы программы, Вы можете получить готовую программу, если будете строго придерживаться указаний по набору Бейсик программы и блоков кодов.

Теперь несколько слов о том, что же может эта программа. Для простоты пользования в программе не применяется полностью графический режим, при помощи которого можно рисовать линии, окружности и т.д. Применяется только псевдографика, позволяющая печатать какой-либо символ в то знакоместо экрана, где находятся курсор. Символьный набор же сделан такой, что при нажатии буквенных клавиш, печатаются всевозможные символы: кружочки (это прежде всего, как в обычной игре "Мозаика"), а также квадратики, треугольнички, крестики и т.д., (насколько хватает фантазии) по количеству буквенных клавиш.

Изображение строится, например, из кружочков, как в игре "Мозаика" из круглых фишек, но в отличие от прототипа, можно использовать "фишки" разной формы и к тому же не беспокоиться о том, что фишек нужного цвета не хватит для желаемого рисунка.

Символьный набор для игры построен таким образом, что переключившись в режим CAPS LOCK, можно печатать буквы русского алфавита. Этот режим нужен не для того, чтобы использовать программу в качестве текстового редактора, а для того, чтобы можно было подписать рисунок.

Для перемещения курсора по экрану используются функциональные клавиши перемещения курсора, а при помощи других функциональных клавиш производится переключение цвета чернил, бумаги, бордюра, включение и выключение режимов повышенной яркости и инверсии. При распределении этих функции между клавишами, ориентир был сделан на удобство пользования для клавиатуры "SPECTRUM+" или что то же самое, отечественного компьютера "ДЕЛЬТА-С". (Хотя это распределение Вы можете изменить на свой вкус.)

Краткая инструкция по пользованию программой всегда находится в памяти компьютера и в любой момент может быть вызвана на экран при нажатии BREAK, при этом изображение с экрана запоминается и после просмотра инструкции возвращается обратно на экран.

В программе предусмотрена возможность сохранения рисунка на внешнем носителе (магнитная лента, дискета) и загрузки рисунка с внешнего носителя. Предусмотрена также возможность оперативного сохранения рисунка в специальном буфере в памяти компьютера. С этим буфером можно производить следующие манипуляции: переслать рисунок с экрана в буфер, вызвать рисунок из буфера на экран, а также обменять содержимое экрана и буфера между собой.

После того, как работа над рисунком закончена, можно насладиться результатами своего труда под музыкальное сопровождение компьютера, нажав ENTER.

Для дальнейшей работы, при необходимости, может быть произведена очистка экрана.

Теперь, когда мы кратко ознакомились с возможностями программы "МОЗАИКА", можно переводить к подробному описанию программы. Она набрана на Бейсике, но в ней используются фрагменты в машинных кодах, в частности те, о которых говорилось в "ZX-РЕВЮ", см. выше.

Учитывая тот факт, что у многих пользователей имеется "БЕТА-ДИСК интерфейс", программа построена таким образом, что адаптация ее под "БЕТА-ДИСК" сводится к минимуму. Об этом будет подробный разговор в конце статьи.

Кроме Бейсик-файла и файла в машинных кодах для работы программы необходимо подготовить например, при помощи "ARTSTUDIO", заставку-картинку, на которой будет содержаться краткая инструкция по клавишам управления программой. Форма, цвет, размер шрифта, окружающее оформление - произвольное, пусть его Вам подскажет Ваша фантазия, но там кроме названия программы должна быть следующая информация:

КЛАВИШИ УПРАВЛЕНИЯ:

[BREAK] -Вызов инструкции
[E.MODE] -Смена цвета чернил
[GRAPH] -Изменение яркости
[EDIT] -Смена цвета бумаги
[INV.V.] -Инверсия вкл./выкл.
[TR. V.] -Смена цвета бордюра
[DELETE] -Забой символа
[ENTER] -Просмотр рисунка
[C.LOCK] -Режим графика/текст

Буквенные клавиши - граф.символы
[CS]+буквенные клавиши, а также
[SS]+2,0,Y,U,D - русские буквы

STOP [SS]+A - Очистка экрана
<= [SS]+Q - Запоминание экрана
> = [SS]+E - Экран из буфера
<> [SS]+W - Обмен экран/буфер
[SS]+I - Загрузка экрана
[SS]+S - Запись экрана

Подготовив файл с этой картинкой-инструкцией, запишите его на магнитофон под именем "mozaik $" CODE 16384,6912.

Кроме этого надо подготовить еще один кодовый файл для работы программы. Это будет "mozaik" CODE 63824,1584. Сюда входят следующие коды:

Во-первых, это символьный набор, содержащий русские буквы и графические образы, причем кодам с 64 по 95 (заглавные латинские буквы) соответствуют русские буквы, а кодам с 96 по 127 (строчные латинские буквы) соответствуют графические символы. Русские буквы в символьном наборе следует расположить в соответствии с тем, как подписана Ваша клавиатура. Можно, например, в соответствии с кодами КОИ-7: "ЯВЕРТЫ...", а можно в соответствии с клавиатурой пишущей машинки: "ЙЦУКЕН...". Поскольку программа не претендует на стыковку с принтером в литерном режиме, то возможен любой вариант. Из-за того, что на "Спектруме" меньше клавишей, чем на пишущей машинке, часто применяется вариант со смещенным верхним рядом "ЦУКЕНГ...". В общем, как Вам нравится, так и делайте. Для примера ниже приведен символьный набор в кодах КОИ-7 (имеется в виду только соответствие заглавным латинским буквам, так как на месте строчных букв находятся графические символы "МОЗАИКИ"). Переставить русские буквы в соответствии с тем, как отмаркированы клавиши Вашего компьютера очень просто при помощи программы "ARTSTUDIO". В ней для этого предусмотрена специальная возможность. Надо загрузить предлагаемый символьный набор в "ARTSTUDIO", затем в текстовом режиме, начиная из левого верхнего угла, напечатать 32 русские буквы в соответствии с требуемым новым порядком расположения, а затем, задав окно вокруг этих напечатанных букв скопировать их с экрана в символьный набор, начиная с символа "@", пользуясь операцией CAPTURE FONT меню FONT EDITOR.

Сразу же за символьным набором следует блок UDG-графики. Правда, используется в программе всего лишь один символ UDG - "A", однако нет гарантии, что в дальнейшем программа не будет усовершенствоваться, поэтому зарезервирован полный объем UDG-блока, 168 байт.

Кроме символьного набора и UDG-блока в кодовом файле расположен блок кодов "ON ERROR GO TO" (см. "ZX-РЕВЮ" N 5-6 за 1992 г. стр. 113). Затем идут два небольших блока, длиной по 32 байта каждый, воспроизводящие звуковые эффекты, (см. "ZX-РЕВЮ" N 11-12

за 1992 г. стр. 241).

Далее расположено несколько блоков кодов, осуществляющих переброску из экрана в память или из памяти на экран. Для простой переброски используется команда LDIR, и тогда процедура переброски будет выглядеть следующим образом:

```
LD    HL,16384     адрес экрана
LD    BC,6912      длина кодов
LD    DE,50000     адрес буфера
LDIR
RET
```

Такой блок кодов имеет длину 12 байт и может быть загружен в любое место памяти. Разумеется, при переброске экрана в буфер, содержимое последнего будет уничтожено, как и содержимое экрана будет уничтожено при переброске буфера на экран. Для того, чтобы обменять содержимое экрана и буфера без уничтожения обоих, применяется другая процедура. Она взята из программы "SUPERCODE" (N 27):

```
LD    DE,#C350     (1)
LD    HL,#4000
LD    B,#1B        (2)
PUSH  BC           (5)
LD    B,#00        (4)
LD    A,(HL)       (5)
PUSH  AF
LD    A,(DE)       (6)
LD    (HL),A       (7)
POP   AF           (8)
LD    (DE),A
INC   HL           (9)
INC   DE
DJNZ  #AB0B        (10)
POP   BC           (11)
DJNZ  #AB08        (12)
RET
```

Блок кодов имеет длину 25 байт. Его можно загружать в любое место памяти. Адрес загрузки является адресом старта. Адрес начала дисплейного файла экрана задан в регистре HL, а начало буфера - в регистре DE (1). Основу блока составляет процедура обмена содержимого одной пары ячеек экрана и буфера (5)-(9). Сначала в аккумулятор загружается содержимое ячейки экрана (5) и отправляется для хранения на стек. Затем в аккумулятор загружается содержимое соответствующей ячейки буфера (6) и из него перегружается в ячейку экрана (7). После этого в аккумулятор со стека считывается запомненное значение экранной ячейки (8) и записывается в буфер. После завершения обмена ячеек содержимое регистров HL и DE увеличиваются на единицу (9) для перехода к следующей паре ячеек.

Для полной переброски дисплейного файла вместе с атрибутами надо эту последовательность действий повторить 6912 раз. Это осуществляется двумя циклами (один внутри другого). Внутренний цикл (4)-(10) организован с максимально возможным числом повторений (в счетчик цикла - регистр B записывается 0, что соответствует 256 повторениям). Внешний цикл должен содержать, таким образом 5912/256 -27 (или #1B) повторений. Внешний цикл - (2)-(12). Так как и во внутреннем и во внешнем циклах счетчиком числа повторений является регистр B, то для того, чтобы значение регистра B не терялось при переходе от внешнего цикла ко внутреннему, оно запоминается на стеке (3) и снимается для контроля при переходе к внешнему циклу (11).

Далее - возврат в вызывающую программу.

Перед первым запуском этого блока кодов необходимо позаботиться о том, чтобы в буфере находились атрибуты белого экрана. Иначе, если буфер содержал, например, нули, то после переброски экран станет черный.

При однократном выполнении этого блока кодов происходит смена изображения экрана и буфера. При следующем выполнении восстанавливаются прежние изображения. Это используется, например, для вызова на экран, инструкции и сохранения при этом рисунка.

Для организации разнообразных обменов между экраном и буферами есть два варианта. Первый: иметь один блок кодов для простой переброски и один блок кодов для обмена экрана и буфера. А перед выполнением требуемой переброски подставлять в нужные ячейки памяти (при помощи Бейсика) соответствующие числа, задавая параметры HL и DE. Но учитывая небольшую длину кодовых блоков, их можно заготовить такое количество, какое потребуется для всевозможных перебросок (подготовка исходных параметров при помощи Бейсика съест больше памяти). Поэтому в программе используются несколько блоков для переброски.

Еще в кодовом файле "mozaik" CODE находится машиннокодовый музыкальный фрагмент, воспроизводящий знакомую мелодию. Он подготовлен при помощи музыкального редактора "WHAM".

Для того, чтобы Вы могли полностью воспроизвести у себя программу "МОЗАИКА", ниже приведены все используемые в программе блоки кодов.

Рассмотрим распределение памяти при работе программы "МОЗАИКА ". RAMTOP при старте программы устанавливается равным 49999.

Буфер, зарезервированный для запоминания экрана 6912 байт	50000
Буфер, зарезервированный для хранения титульной заставки (инструкции) 6912 байт	56912
Символьный набор 768 байт	63824
Блок UDG-графики 168 байт	64592
Блок ON ERROR GO TO 73 байта	64760
Звук SOUND 1 32 байта	64833
Звук SOUND 2 32 байта	64865
Процедура для переброски экрана в буфер 12 байт	64897
Процедура для переброски буфера на экран 12 байт	64909
Процедура для обмена экрана и буфера 25 байт	64921
Процедура для переброски инструкции (заставки) из памяти на экран 12 байт	64946
Обмен экрана и заставки 25 байт	64958
Музыкальный фрагмент 525 байт	64983
	65508

Теперь, когда мы разобрались в общих чертах с кодами, можно приступить к описанию Бейсиковского файла программы "МОЗАИКА". Текстовые сообщения на русском языке в листинге программы напечатаны по-русски, хотя так на самом деле не выглядят. Это сделано только для удобочитаемости программы. По-русски они будут выглядеть после переключения символьного набора при работе программы.

```
1 GO TO 100
2 BORDER 7: PAPER 7: INK 0: CLEAR 49999: POKE 23739,111: LOAD "mozaik $"CODE 16364
3 LOAD "mozaik"CODE 63834
4 RANDOMIZE USR 64946: GO SUB 7: GO SUB 20: CLS : RANDOMIZE USR 64897: RUN
5 SAVE "MOZAIK" LINE 2: SAVE "mozaik $"CODE 56912,6912: SAVE "mozaik"CODE 63824,1684
6 VERIFY "MOZAIK": VERIFY "mozaik $"CODE : VERIFY "mozaik"CODE : GO TO 9999
7 POKE 23675,80: POKE 23676,252: RETURN : REM udg
8 POKE 23606,80: POKE 23607,248: RETURN : REM rus
9 POKE 23606,0: POKE 23607,50: RETURN : REM lat
10 POKE 64813,PEEK 23670: POKE 64813,PEEK 23671: RANDOMIZE USR 64760: RETURN : REM err
20 GO SUB 8: PRINT AT 1,0; INK 7; PAPER 1; BRIGHT 1; FLASH 1;"     нажмите любую клавишу
     "
22 BEEP .1,26: BEEP .1,20
24 RANDOMIZE 29: GO SUB 10: PAUSE 0
29 RETURN
30 BORDER 7: INPUT ;: PRINT PAPER 7; INVERSE 0; AT 0,0;"                                  "
39 RETURN
40 IF x>31 THEN LET x=0: LET y=y+1
42 IF x<0 THEN LET x=31: LET y=y-1
44 IF y>21 THEN  LET y=1
46 IF y<1 THEN  LET y=21
49 RETURN
50 GO SUB 30
52 PRINT AT 0,0:" введите имя (макс.10 символов) "
54 GO SUB 9: INPUT LINE f$: IF f$="" THEN LET f$="$"
56 GO SUB 30
59 RETURN
60 GO SUB 8: PRINT AT 0,0; INK 7; PAPER 2; BRIGHT 1; FLASH 1; "            ошибка !
     ": BEEP 1,0: PAUSE 0
69 RETURN
70 RANDOMIZE 60: GO SUB 10: LOAD f$ CODE 16384
79 RETURN
80 RANDOMIZE 60: GO SUB 10: SAVE f$ CODE 16384,6912
89 RETURN
100 LET y=20: LET x=3: LET bb=3: LET p=7: LET i=2: LET b=1: LET f=0: LET v=0
1000 RANDOMIZE 1000: GO SUB 10:BORDER bb: INPUT ;: GO SUB 8
1010 PRINT PAPER 7; INK 0; BRIGHT 0; INVERSE 0;AT 0,0; " чернила-  яркость-  бумага-  "
1020 PRINT INK i; PAPER p; BRIGHT b; INVERSE v; AT 0,10; CHR$ 143 ; AT 0,29;" "; INK 7;AT
     0,20; INVERSE 1;" ";
1100 GO SUB 40
1110 PRINT  OVER 1; PAPER 8; INK 8; BRIGHT 8; FLASH f; INVERSE v;AT y,x;"A";CHR$ 8;: REM A-
     GRAPH
1200 RANDOMIZE 2000: GO SUB 10:PAUSE 0: LET i$=INKEY$
1210 PRINT OVER 1; PAPER 8; INK 8; BRIGHT 8; FLASH 0; INVERSE v; "A";CHR$ 8;: REM A-GRAPH
1300 RANDOMIZE 1000: GO SUB 10
1310 IF CODE INKEY$=8 THEN LET x=x-1: GO TO 1100
1320 IF CODE INKEY$=9 THEN LET x=x+1: GO TO 1100
1330 IF CODE INKEY$=10 THEN LET y=y+1: GO TO 1100
1340 IF CODE INKEY$=11 THEN LET y=y-1: GO TO 1100
1350 IF CODE INKEY$=12 THEN LET x=x-f: GO SUB 40: PRINT AT y,x;" ": RANDOMIZE USR 64865: GO
     TO 1100
1360 IF INKEY$=" " THEN PRINT AT y,x;" ": RANDOMIZE USR 64833:LET x=x+1: GO TO 1100
1365 IF INKEY$="!" AND p+1=0 THEN GO TO 9999
1370 IF i$=" AND " THEN  LET i$="[": REM TOKEN
1380 IF i$=" STEP " THEN LET i$="\": REM TOKEN
1390 IF i$=" OR " THEN LET i$="]": REM TOKEN
1400 IF i$=" " AND i$<="z" THEN PRINT PAPER p; INK i; BRIGHT b; INVERSE v;i$;: RANDOMIZE USR
     64833: LET x=x+f: GO TO 1100
1410 IF i$=" NOT " THEN GO SUB 50: GO SUB 80: REM TOKEN
```

```
1420 IF i$=" AT " THEN GO SUB 50: GO SUB 70: REM TOKEN
1430 IF i$="<=" THEN GO SUB 30: RANDOMIZE USR 64897: PRINT AT 0,0;
      "          экран в памяти          ": BEEP 0.1,26: PAUSE 50
1440 IF i$=">=" THEN GO SUB 30:RANDOMIZE USR 64909: BEEP 0.1,20
1450 IF i$="<>" THEN GO SUB 30: RANDOMIZE USR 64921: BEEP 0.1,26: BEEP 0.1,20
1460 IF i$=" STOP " THEN CLS
1470 IF CODE i$=15 THEN LET b=b=0
1480 IF CODE i$=5 THEH LET v=v=0
1490 IF CODE i$=6 THEN LET f=f=0: POKE 23658,8*(2-f)
1500 IF CODE i$=14 THEN LET i=(1+1)*(i<7)
1510 IF CODE i$=7 THEN LET p=(p+1)*(p<7)
1520 IF CODE i$=4 THEN LET bb=(bb+1)*(bb<7)
1530 IF CODE i$=13 THEN GO SUB 30: FOR c=0 TO 50: NEXT c: RANDOMIZE USR 64983
1999 GO TO 1000
2000 BORDER 7: INPUT ;
2010 RANDOMIZE USR 64958: BEEP .1,36: BEEP .1,20
2020 RANDOMIZE 2030: GO SUB 10:PAUSE 0
2030 RANDOMIZE USE 64958: BEEP .1,26: BEEP .1,20
2040 BORDER bb: INPUT ;: GO TO 1200
9999 GO SUB 9: BORDER 7: POKE 23658,16
```

Переменные, применяемые в программе:
y - вертикальная координата курсора (строка)
x - горизонтальная координата курсора (колонка)
bb - цвет бордюра
p - цвет бумаги (PAPER)
i - цвет чернил (INK)
b - яркость (BRIGHT)
f - мигание (FLASH)
v - инверсия (INVERSE)
i$ - нажатая клавиша

Автостарт программы происходит со строки 2. Здесь задаются цвета экрана и резервируется место для работы блока кодов программы. Кроме этого, при помощи POKE отменяется вывод на экран сообщений типа: "Bytes:" при загрузке блоков кодов. Первый загружаемый блок - это заставка-инструкция, и благодаря этому POKE при загрузке следующего кодового файла (в строке 3) экран не будет испорчен. Этот прием уже был описан на страницах "РЕВЮ".

В строке 4 заставка-инструкция при помощи блока машинных кодов (смотри распределение памяти выше) перебрасывается в буфер для хранения инструкции (в адрес 56912). Далее подпрограмма GO SUB 7 производит переключение блока UDG на загруженные коды. Выполняется подпрограмма GO SUB 20, выводящая на экран табличку "нажмите любую клавишу" (место для вывода таблички назначьте такое, чтобы она не уничтожала информации на экране). Звуковой сигнал говорит о том, что загрузка программы завершена и она ждет команды начала работы. Программа останавливается на PAUSE 0 в строке 24, а непосредственно перед паузой конструкция: RANDOMIZE 29: GO SUB 10 задает параметр для работы кодового блока "ON ERROR GO TO", что означает переход в случае ошибки (нажатии BREAK) на строку с номером 29. То есть при нажатии любой клавиши, в том числе и BREAK, будет выполнен возврат на строку 4. При этом происходит следующее. Очищается экран и в этот момент содержимое этого чистого экрана засылается в буфер экрана (в адрес 50000), подготавливая таким образом атрибуты изображения в буфере в случае последующего обмена экрана и буфера, чтобы экран после обмена не выглядел черным. Теперь подготовка завершена и происходят старт программы точно так же, как Вы будете это делать после остановки программы - командой RUN.

По команде RUN происходит старт программы с первой строки, которая переадресует на строку 100. В строках до 100 расположены различные подпрограммы; их назначение будет рассмотрено в процессе описания. В строке 100 присваиваются первоначальные значения переменным, используемым в программе. После этого начинается собственно

выполнение основной части программы.

Строка 1000 является базовой, на нее может быть сделан переход из различных мест программы, поэтому в начале этой строки задается переход на нее же в случае ошибки, которая может возникнуть при работе программы. Далее задается цвет бордюра и происходит окрашивание в цвет бордюра двух системных строк экрана. Они для создания рисунка недоступны. Подпрограмма GO SUB 8 производит переключение символьного набора на тот, который загружен, с графическими образами и русскими буквами.

Строки 1010 и 1020 выводят в верхней части экрана индикаторную строку, в которой отображается текущее состояние цветов чернил и бумаги, а также значение яркости. Индикаторная строка также недоступна для создания рисунка.

Строка 1100 также является базовой. На нее также происходят переходы из различных мест программы. На нее замыкается малый цикл опроса клавишей в который входят опрос курсорных клавишей, клавиши забоя, пробела и буквенных клавишей. Поскольку быстродействие Бейсика невелико, такой прием позволил реализовать вполне приемлемое быстродействие при перемещении курсора и печатании символов.

Подпрограмма GO SUB 40 - это контроль положения курсора. При перемещении курсора, при достижении правого края экрана происходит переход на левый край но на следующей строке экрана (строка 40 программы). При достижении левого края - курсор перескакивает на правый край на предыдущую строку экрана (строка 42). При вертикальных перемещениях курсора изменения номера колонки не происходит (строки 44,46).

Строка 1110 выводит на экран курсор. Его изображение закреплено за символом "A" UDG-графики. Цвет курсора определяется теми атрибутами, которые установлены для этого знакоместа. Если там ничего еще не нарисовано, то это белая бумага и черные чернила. Однако, если в этом знакоместе уже напечатан какой либо символ, то цвет курсора будет зависеть от атрибутов этого знакоместа. Кроме того, курсор печатается в режиме OVER 1, поэтому он накладывается на символ, находящийся в этом знакоместе. Вид курсора будет меняться в зависимости от включенного режима инверсии, а также от параметра мигания. Забегая вперед, скажу, что мигание курсора свидетельствует о переключении из режима "графика" в режим "текст". Или иными словами, это включение режима CAPS LOCK.

В строке 1200 происходит ожидание нажатия клавиши. Так как это может быть клавиша BREAK, то на этот случай задается переход на строку 2000, где происходит вывод на экран инструкции. Это делается так. В строке 2000 устанавливается белый цвет бордюра и двух нижних строк экрана, затем (2010) при помощи кодового блока происходит обмен содержимого экрана и буфера инструкции, затем звуковой сигнал и пауза для ожидания нажатия любой клавиши (2020). Но непосредственно перед паузой - установка блока ON ERROR GO TO на следующую после паузы строку. Теперь при нажатии на любую клавишу, в том числе и на клавишу BREAK, произойдет переход на строку 2030, то есть будет продолжено выполнение программы. Еще раз происходит обращение к кодовому блоку, в результате чего происходят восстановление рисунка на экране, а инструкция отправляется назад в свой буфер. Далее - звуковой сигнал, восстановление цвета бордюра и двух нижних строк экрана. Затем возврат в то место программы, откуда был сделан переход на 2000-ю строку, то есть на строку 1200.

Если в строке 1200 нажата не клавиша BREAK, то продолжается выполнение программы со строки 1210. Здесь еще раз происходит печать курсора в режиме OVER 1, в результате чего восстанавливается прежнее изображение в этом знакоместе.

Со строки 1300 начинается определение действий в зависимости от нажатой клавиши. Строки 1310-1340 определяют перемещение курсора.

Строка 1350 - удаление символа в текущем знакоместе. Причем в зависимости от того, в каком режиме находится программа: в графическом или в текстовом, забой действует по-разному. В графическом режиме удаляется символ в позиции курсора, а в текстовом - символ слева от курсора, так привычнее при печати текста. Текущий режим "графика" или "текст" определяется по миганию курсора. Если показатель мигания f=0, то режим "графика", а если f=1, то режим "текст". В зависимости от этого происходит или нет изменение текущей координаты "x" курсора. Удаление символа происходит путем печати

пробела в позиции с текущими координатами, что сопровождается соответствующим звуком.

Строка 1360 реагирует на нажатие пробела. В этом случае, как и при удалении символа, происходит печать пробела в знакоместе с текущими координатами, что также приводит к удалению прежнего символа, однако независимо от режима (мигающий курсор или нет) происходит увеличение на единицу горизонтальной координаты курсора. Это может оказаться полезно при "расчистке" небольших участков рисунка в режиме "графика", а в режиме "текст" привычно печатается пробел. (Если Вы считаете, что это слишком запутанно, то просто удалите строку 1360.)

Строка 1365 является "жучком", позволяющим остановить программу для внесения каких-либо изменений в процессе отладки. Для остановки надо ввести символ "!" (SYMBOL SHIFT + 1). Чтобы остановка не происходила самопроизвольно в процессе работы при попытке напечатать этот символ, для остановки программа требуется дополнительное условие: чтобы переменные, определяющие цвета чернил и бумаги имели нулевые значения (черные чернила и черная бумага - то есть ситуация, бессмысленная при обычной работе). В случае соблюдения этих условий программа переходит на строку 9999, в которой происходит переключение на символьный набор ПЗУ Спектрума, установка белого бордюра, а также принудительное отключение регистра CAPS LOCK для удобства редактирования.

Строки 1370-1390 нужны для того, чтобы можно было печатать символы, которые обычно можно напечатать в расширенном режиме EXTEND MODE. В символьном наборе КОИ-7 русские буквы "Ш" и "Щ" расположены на месте "[" и "]", а последние набираются в режиме EXT.MODE. Поэтому используется прием, позволяющий вводить эти символы при помощи SYMBOL SHIFT вместо "AND" и "OR" соответственно. Аналогично вводится "Э" вместо "STEP", так как эта буква находится на месте символа "\". (Этот прием, кстати, позаимствован из "АРТСТУДИИ".) Если же Вы будете использовать символьный набор с другим расположением русских букв, то продумайте вопрос набора тех букв, которые не помещаются на клавишах с 26-ю латинскими буквами.

Строка 1400 выполняет печать графических символов и русских букв. Печать сопровождается специфическим звуком, воспроизводимым блоком в кодах. В том случае, если программа находится в режиме "текст", то очевидно, что после печати очередного символа, курсор должен переместиться на следующее знакоместо. При построении рисунка по-видимому такого перемещения не должно быть, так как заранее не известно, в какую сторону должен быть сдвинут курсор для построения рисунка (может быть, вверх или вниз). Поэтому в режиме "графика" перемещения курсора не происходит. Режим определяется по миганию курсора (параметр f). Вопрос о том, должен ли курсор перемещаться или нет является в общем-то спорным, поэтому, если Вы хотите, чтобы курсор всегда после печати перемещался на одну позицию вправо, то в строке 1400 вместо: x=x+f подставьте: x=x+1.

Строки, связанные с нажатием клавишей, с 1300 по 1400 заканчиваются все одинаково: зацикливанием на строку 1100. Это, так называемый, малый цикл опроса клавишей. Это, как уже говорилось, сделано для ускорения реагирования программы на нажатие основных клавишей. Со строки 1410 начинается расширенный цикл опроса. В конце этих строк нет перехода на базовую строку. Переход происходит а самом конце расширенного цикла: в строке 1999 программа зацикливается на строку 1000.

Строка 1410 обеспечивает запись экрана на магнитную ленту при нажатии SYMBOL SHIFT+"S" (на клавише ключевое слово SAVE). Вначале производится ввод имени файла (GO SUB 50). Вы можете не задавать никакого имени, а просто нажать ENTER, тогда файл будет записан с именем "$". Можно отменить ввод имени и вернуться к работе программы, если при вводе имени нажать "КУРСОР ВНИЗ". (В обычной Бейсик-программе в таком случае происходит остановка программы по ошибке, но в программе с блоком ON ERROR GO TO происходит переход на заданную строку. Строка для перехода по ошибке была задана последний раз в строке 1300, то есть произойдет переход на строку 1000.) После ввода имени выполняется подпрограмма записи GO SUB 80. В этой подпрограмме блок ON ERROR GO TO устанавливается на строку 60. Это позволит Вам прервать в случае необходимости запись и вернуться к работе через подпрограмму с адреса 60. Это вывод на экран таблички

"ОШИБКА" и звуковой сигнал.

RETURN в конце этой подпрограммы вернет на строку 1420.

Строка 1420 обеспечивает загрузку экрана с магнитной ленты при нажатии SYMBOL SHIFT + "I" (ключевое слово INPUT). Привычнее, наверное, был бы вариант с клавишей "L", или "J" (ключевое слово LOAD), однако эти варианты соответствуют символам, печатаемым в текстовом режиме: "=" и "-". Если все же Вы считаете, что клавиша "I" неудобна, то можно попробовать компромиссный вариант. Вместо 1420 надо будет новую строку расположить внутри малого цикла опроса клавишей:

```
1395 IF i$="=" AND f=0 THEN GO SUB 50: GO SUB 70: GO TO 1000
```

При этом, правда, загрузка будет возможна только в режиме "графика". Зато в режиме "текст" сохранится возможность печатать "=". Выбирайте, что удобнее.

Строка 1430 позволяет сохранять экран в специальном буфере (с адреса 50000). Переброска производится при помощи машиннокодового блока. В процессе работы над рисунком полезно периодически делать запоминание экрана. Строка 1440 производит противоположное действие - вызов экрана из памяти. Например, после каких-то изменений в рисунке, захочется восстановить предыдущий вариант. Строка 1450 производит обмен экрана и буфера без потери изображения обоих. Есть возможность сравнить два варианта рисунка и выбрать лучший.

Строка 1460 выполняет очистку экрана при нажатии SYMBOL SHIFT + "A" (ключевое слово STOP).

Строки 1470-1520 позволяют изменять яркость, инверсию, переключать режим "графика" - "текст", изменять цвет чернил, бумаги и бордюра. Для наглядности выражение типа b=b=0 можно было бы записать так: b=(b=0). То, что стоит в скобках, является логическим выражением, оно равно единице, если b=0. Таким способом достигается переключательный эффект изменения яркости, инверсии и режима "графика" - "текст" при нажатии соответствующих клавиш. При переключении цвета чернил, бумаги и бордюра ситуация похожа. При нажатии на клавишу, цвет (например чернил - 1) изменяется от 0 до 7, а при следующем нажатии (i=8) логическое выражение (i<7) станет равно 0, что вызывает повторение цикла изменения цвета, начиная с i=0.

Строка 1530 воспроизводит "музыкальное сопровождение". Циклическое проигрывание мелодии происходит до тех пор, пока не будет нажата какая-нибудь клавиша. Поэтому, если чуть-чуть задержать руку на клавише ENTER, то мелодия прервется на первом же звуке. Для устранения этого недостатка введен замедляющий пустой цикл FOR-NEXT (в отличие от PAUSE, которая в данном случае не поможет, так как прекращается как раз при нажатии клавиши).

Строка 1999 завершает цикл опроса клавишей, передавая управление на базовую строку 1000. С этого момента работа программы повторяется.

Теперь, когда мы покончили с Бейсиком, можно переходить к конкретным действиям по набору программы. Все коды, которые требуются для работы "МОЗАИКИ", (то есть то, что расположено с адреса 63824 и входит в файл "mozaik" CODE), приведены ниже. Число в конце каждой строки после двоеточия - это контрольная сумма строки. Она поможет Вам при наборе этого блока кодов.

```
F950:00 00 00 00 00 00 00 00 :49
F958:00 00 10 10 10 00 10 00 :91
F960:00 00 28 28 00 00 00 00 :A9
F968:00 00 28 7C 28 7C 28 00 :D1
F970:00 10 3C 50 38 14 78 10 :D9
F978:00 00 64 68 10 2C 4C 00 :C5
F980:00 10 28 10 34 48 34 00 :71
F988:00 00 08 10 00 00 00 00 :99
F990:00 00 08 10 10 10 08 00 :C9
F998:00 00 20 10 10 10 20 00 :01
F9A0:00 00 28 10 7C 10 28 00 :85
F9A8:00 00 10 10 7C 10 10 00 :5D
F9B0:00 00 00 00 00 10 10 20 :E9
F9B8:00 00 00 00 7C 00 00 00 :2D
F9C0:00 00 00 00 00 30 30 00 :19
F9C8:00 00 04 08 10 20 40 00 :3D
F9D0:00 00 38 4C 54 64 38 00 :3D
F9D8:00 00 10 30 10 10 38 00 :69
F9E0:00 00 38 04 18 20 3C 00 :89
F9E8:00 00 38 04 18 04 38 00 :71
F9F0:00 00 18 28 48 7C 08 00 :F5
F9F8:00 00 38 20 38 04 38 00 :BD
FA00:00 00 38 40 78 44 38 00 :66
FA08:00 00 3C 04 08 10 10 00 :6A
FA10:00 00 38 44 38 44 38 00 :3A
FA18:00 00 38 44 3C 04 38 00 :06
FA20:00 00 00 10 00 00 10 00 :3A
FA28:00 00 00 10 00 00 10 20 :62
```

```
FA30:00 00 08 10 20 10 08 00 :7A
FA38:00 00 00 7C 00 7C 00 00 :2A
FA40:00 00 20 10 08 10 20 00 :A2
FA48:00 00 18 24 08 00 08 00 :8E
FA50:00 00 4C 52 72 52 4C 00 :F8
FA58:00 00 38 44 7C 44 44 00 :D2
FA60:00 00 78 40 78 44 78 00 :46
FA68:00 00 48 48 48 48 7C 04 :02
FA70:00 00 1C 24 24 24 7E 42 :B2
FA78:00 00 7C 40 78 40 7C 00 :62
FA80:00 00 7C 54 54 7C 10 00 :2A
FA88:00 00 3C 20 20 20 20 00 :3E
FA90:00 00 44 28 10 28 44 00 :72
FA98:00 00 44 4C 54 64 44 00 :1E
FAA0:00 10 44 4C 54 64 44 00 :36
FAA8:00 00 44 48 70 48 44 00 :2A
FAB0:00 00 1C 24 24 24 44 00 :76
FAB8:00 00 44 6C 54 44 44 00 :3E
FAC0:00 00 44 44 7C 44 44 00 :46
FAC8:00 00 38 44 44 44 38 00 :FE
FAD0:00 00 7C 44 44 44 44 00 :56
FAD8:00 00 3C 44 3C 24 44 00 :F6
FAE0:00 00 78 44 44 78 40 00 :92
FAE8:00 00 38 44 40 44 38 00 :1A
FAF0:00 00 7C 10 10 10 10 00 :A6
FAF8:00 00 44 44 3C 04 38 00 :F2
FB00:00 00 54 54 38 54 54 00 :83
FB08:00 00 78 44 78 44 78 00 :F3
FB10:00 00 40 40 78 44 78 00 :BF
FB18:00 00 42 42 72 4A 72 00 :C5
FB20:00 00 38 44 08 44 38 00 :1B
FB28:00 00 44 54 54 54 7C 00 :DF
FB30:00 00 38 04 1C 04 38 00 :BF
FB38:00 00 44 54 54 54 7E 02 :F3
FB40:00 00 44 44 3C 04 04 00 :07
FB48:00 00 60 20 38 24 38 00 :57
FB50:00 00 18 24 70 20 7C 00 :93
FB58:FF FF FF 7F 7F 3F 1F 07 :B3
FB60:00 00 00 FF FF 00 00 00 :59
FB68:AA 55 AA 55 AA 55 AA 55 :5F
FB70:FF 7F 3F 1F 0F 07 03 01 :61
FD78:01 01 07 0F 1F 3F 7F FF :69
FB80:FF FE FC F8 F0 E0 C0 80 :7C
FB88:03 07 0F 0F 0F 0F 07 03 :D3
FB90:FF FF 7E 3C 00 00 00 00 :43
FB98:00 00 00 F8 F8 18 18 18 :CB
FBA0:18 18 18 1F 1F 00 00 00 :21
FBA8:18 18 18 F8 F8 00 00 00 :DB
FBB0:18 18 18 FF FF 00 00 00 :F1
FBB8:18 18 18 F8 F8 18 18 18 :33
FBC0:18 18 18 1F 1F 18 18 18 :89
FBC8:00 00 00 FF FF 18 18 18 :09
FBD0:18 18 18 FF FF 18 18 18 :59
FBD8:07 1F 3F 7F 7F FF FF FF :33
FBE0:80 C0 E0 F0 F8 FC FE FF :DC
FBE8:FF FF FF FE FE FC F8 E0 :B0
FBF0:00 00 00 00 3C 7E FF FF :A3
FBF8:00 00 00 1F 1F 18 18 18 :79
FC00:18 18 18 18 18 18 18 18 :BC
FC08:E0 F8 FC FE FE FF FF FF :D1
FC10:FF FF FF FF FF FF FF FF :04
FC18:C0 E0 F0 F0 F0 F0 E0 C0 :14
FC20:3C 7E FF FF FF FF 7E 3C :8C
FC28:00 0E 08 30 08 08 0E 00 :88
FC30:00 08 08 08 08 08 08 00 :5C
FC38:00 70 10 0C 10 10 70 00 :50
FC40:00 14 28 00 00 00 00 00 :78
FC48:3C 42 99 A1 A1 99 42 3C :B4
FC50:FF 81 81 99 99 81 81 FF :80
FC58:00 7C 42 7C 42 42 7C 00 :8E
FC60:00 3C 42 40 40 42 3C 00 :D8
FC68:00 78 44 42 42 44 78 00 :60
FC70:00 7E 40 7C 40 40 7E 00 :A4
FC78:00 7E 40 7C 40 40 40 00 :6E
FC80:00 3C 42 40 4E 42 3C 00 :06
FC88:00 42 42 7E 42 42 42 00 :4C
FC90:00 3E 08 08 08 08 3E 00 :28
FC98:00 02 02 02 42 42 3C 00 :5A
FCA0:00 44 48 70 48 44 42 00 :66
FCA8:00 40 40 40 40 40 7E 00 :62
FCB0:00 42 66 5A 42 42 42 00 :74
FCB8:00 42 62 52 4A 46 42 00 :7C
FCC0:00 3C 42 42 42 42 3C 00 :3C
FCC8:00 7C 42 42 7C 40 40 00 :C0
FCD0:00 3C 42 42 52 4A 3C 00 :64
FCD8:00 7C 42 42 7C 44 42 00 :D6
FCE0:00 3C 40 3C 02 42 3C 00 :14
FCE8:00 FE 10 10 10 10 10 00 :32
FCF0:00 42 42 42 42 42 3C 00 :72
FCF8:CD 7C 00 3B 3B E1 01 0F :A4
FD00:00 09 EB 2A 3D 5C 73 23 :4A
FD08:72 C9 3B 3B CD 8E 02 7B :8E
FD10:FE FF 20 F8 3A 3A 5C FE :F0
FD18:FF 28 21 FE 07 28 1D FE :A5
FD20:08 28 19 3C 32 81 5C FD :AE
FD28:36 00 FF 21 D0 07 22 42 :B6
FD30:5C AF 32 44 5C FD CB 01 :D3
FD38:FE C3 7D 1B 33 33 C3 03 :BA
FD40:13 0E 01 06 28 21 64 00 :12
FD48:C5 11 01 00 E5 CD B5 03 :86
FD50:E1 11 1E 00 ED 5A C1 10 :75
FD58:EF 3E 02 0C 41 B8 20 E3 :8C
FD60:C9 0E 01 06 28 21 E8 03 :6F
FD68:C5 11 01 00 E5 CD B5 03 :A6
FD70:E1 11 18 00 ED 52 C1 10 :87
FD78:EF 3E 02 0C 41 B8 20 E3 :AC
FD80:C9 21 00 40 01 00 1B 11 :D4
FD88:50 C3 ED B0 C9 21 50 C3 :32
FD90:01 00 1B 11 00 40 ED B0 :97
FD98:C9 11 50 C3 21 00 40 06 :E9
FDA0:1B C5 06 00 7E F5 1A 77 :87
FDA8:F1 12 23 13 10 F6 C1 10 :B5
FDB0:F0 C9 21 00 40 01 00 1B :E3
FDB8:11 50 DE ED B0 C9 11 50 :BB
FDC0:DE 21 00 40 06 1B C5 06 :E8
FDC8:00 7E F5 1A 77 F1 12 23 :EF
FDD0:13 10 F6 C1 10 F0 C9 21 :91
FDD8:E0 FE 22 F2 FD 21 62 FF :46
FDE0:22 F6 FD F3 CD 1E FE CD :9B
FDE8:8E 02 1C 28 F7 FB C9 0A :7E
FDF0:12 07 EF FE E1 FE 71 FF :42
FDF8:63 FF EB 5E 23 56 13 1A :46
FE00:FE 40 28 12 72 2B 73 C9 :4F
FE08:7E C6 0C 5F 16 00 21 AB :97
FE10:FE 19 66 2E 01 C9 23 5E :04
FE18:23 56 2B 2B 18 E1 21 F2 :F1
FE20:FD CD FB FD 32 EF FD 21 :1F
FE28:F6 FD CD FB FD 32 F0 FD :FD
FE30:21 EF FD CD 08 FE CB 13 :EC
FE38:DA E1 FE E5 21 F0 FD CD :AF
```

```
FE40:08 FE D1 7C 3D 20 04 7A :6C
FE48:3D 28 42 3A FA FD 4F 06 :73
FE50:00 3A F1 FD 08 3A F1 FD :A6
FE58:DD 62 16 10 00 00 08 1D :E0
FE60:D3 FE 20 17 DD 5C AA 08 :51
FE68:2D C2 82 FE D3 FE 6C AA :BC
FE70:10 EA 0C C2 5E FE C9 61 :BC
FE78:64 61 6D 28 FE 08 2D CA :CD
FE80:6C FE D3 FE 00 00 10 D4 :9D
FE88:0C C2 5E FE C9 3A FA FD :AA
FE90:2F 4F C5 F5 06 00 E5 21 :D2
FE98:00 00 CB 2E CB 2E CB 2E :81
FEA0:00 E1 10 F2 0D C2 96 FE :E4
FEA8:F1 C1 C9 FF F0 E3 D7 CB :95
FEB0:C0 B4 AB A1 97 90 88 80 :9D
FEB8:79 72 6C 66 60 5B 56 51 :D5
FEC0:4C 48 44 40 3D 39 36 33 :B5
FEC8:30 2D 2B 28 26 24 22 20 :02
FED0:1E 1C 1B 19 18 17 15 14 :94
FED8:13 12 11 10 0F 0E 0D 0C :52
FEE0:01 0F 0D 06 29 0A 29 0A :67
FEE8:29 0F 0D 06 29 0A 29 0A :97
FEF0:29 0F 0D 06 29 0A 29 03 :98
FEF8:29 0A 29 01 29 0B 29 0B :BB
FF00:29 0F 0D 01 29 0B 29 0B :AD
FF08:29 0F 0D 01 29 0B 29 0B :B5
FF10:29 0F 0D 01 29 0B 29 03 :B5
FF18:29 0B 29 06 29 0A 29 0A :E0
FF20:29 29 29 1E 29 12 29 1B :37
FF28:29 12 29 19 29 11 29 11 :18
FF30:29 29 29 01 29 0B 29 03 :0B
FF38:29 0B 29 06 29 0A 29 0A :00
FF40:29 29 29 1E 29 12 29 1B :57
FF48:29 12 29 19 29 11 29 11 :38
FF50:29 29 29 01 29 0B 29 03 :2B
FF58:29 0B 29 06 29 0A 29 0A :20
FF60:29 40 00 29 29 29 29 12 :7E
FF68:29 12 29 29 29 29 29 12 :81
FF70:29 12 29 29 29 29 29 12 :89
FF78:29 29 29 12 29 29 29 11 :90
FF80:29 11 29 29 29 29 29 11 :97
FF88:29 11 29 29 29 29 29 11 :9F
FF90:29 11 29 29 29 29 29 11 :A7
FF98:29 29 29 11 29 29 29 12 :B0
FFA0:29 12 29 29 29 29 29 0A :B1
FFA8:29 29 29 0A 29 29 29 0B :B2
FFB0:29 0B 29 29 29 29 29 11 :C1
FFB8:29 29 29 11 29 29 29 12 :D0
FFC0:29 12 29 29 29 29 29 0A :D1
FFC8:29 29 29 0A 29 29 29 0B :D2
FFD0:29 0B 29 29 29 29 29 11 :E1
FFD8:29 29 29 11 29 29 29 12 :F0
FFE0:29 12 29 40 00 00 00 00 :83
```

Для набора кодового файла "mozaik" CODE 63824,1684 можно использовать любую программу-монитор, которая есть под рукой. Однако для того, чтобы избежать ошибок, которые обязательно возникнут при таком количестве набираемых цифр, удобно будет воспользоваться программой для ввода, опубликованной в "ZX-РЕВЮ" N 3 за 1991 г. на стр. 59 в статье "ZX-MODEM", оформив числовые данные в виде строк DATA. Учитывая то, что не все имеют возможность ознакомиться с этой статьей, мы рекомендуем Вам воспользоваться несколько измененным вариантом программы для ввода кодов. В приведенном ниже варианте данные вводятся непосредственно при помощи INPUT. (Возможно, так удобнее). Кроме того, периодически, вводя "S", Вы имеете возможность сохранить на ленте то, что уже введено. Приведенные выше коды программы "МОЗАИКА" напечатаны вразрядку, да еще с разделителями ":". Это сделано для удобочитаемости данных. Вам при наборе надо будет вводить их подряд, например для первой строки:

```
F950000000000000000049 [ENTER]
```

Никаких пробелов или двоеточий вводить не надо. Ошибки, допущенные при вводе, будут отмечены звуковым сигналом, после чего Вам будет предложено повторить ввод.

```
1 GO TO 100
2 LOAD "mozaik" CODE
4 GO TO 0
5 SAVE "INPUT" LINE 2: STOP
100 BORDER 7: PAPER 7: INK 0: CLEAR 29999
110 DIM a(10)
120 DEF FN A(a$)=(CODE a$(1)-48-(7 AND a$(1)"9"))*16+(CODE a$(2)-48-(7 AND a$(2)>"9"))
1000 POKE 23658,8: INPUT "DATA:"; LINE b$
1010 IF h$ = "S" THEN  GO TO 2000
1020 LET a$=h$
1030 LET sum=0
1040 FOR i=1 TO 2
1050 LET b$=a$(2*i-1 TO 2*i)
1060 LET a(i)=FN A(b$)
1070 NEXT i
1080 LET add=a(1)*256+a(2)
1090 LET sum=a(1)+a(2)
1100 FOR i=3 TO 10
```

```
1110 LET b$=h$(2*i-1 TO 2*i)
1120 LET a(i)=FN A(b$)
1130 LET sum=sum+a(i)
1140 POKE add, a(i)
1150 LET add-add+1
1160 NEXT i
1170 LET b$=h$(21 TO )
1180 LET cs=FN A(b$)
1190 LET cs1=sum-256*INT (sum/256)
1200 IF cs<>cs1 THEN  PRINT a$; INVERSE 1;" ERROR ! ": BEEP 1,0:LET add=add-8: GO TO 1000
1210 PRINT a$( TO 4): GO TO 1000
2000 SAVE "mozaik" CODE 63824,1684
2010 VERIFY "mozaik"CODE
2020 CLS : PRINT a$( TO 4): GO TO 1000
```

Набрав эту Бейсик-программу, сделайте RUN 5 для записи ее на магнитную ленту (автостарт со 2-й строки обеспечит в дальнейшем автоматическую подгрузку уже частично набранного кодового файла). После этого запустите программу: RUN и начинайте ввод кодового файла "mozaik" CODE. При перерывах в работе сохраняйте его на магнитной ленте, вводя "S".

Теперь последовательность, в которой следует набирать всю программу "МОЗАИКА". Сначала надо заготовить картинку-заставку при помощи графического редактора. Затем набрать приведенный выше кодовый файл. Теперь можно приступить к набору Бейсик-файла. Набирая 10 строку, подставьте в ее начало RETURN: это отключит пока блок кодов ON ERROR GO TO, блокирующий остановку программы. В строке 30 в кавычках набрано 32 пробела. Набирая текст на русском языке, учитывайте, какой русский символьный набор будет применяться в Вашей программе. Можно рекомендовать такой подход. Загрузите уже набранный кодовый файл "mozaik" CODE, затем, непосредственно перед набором русского текста, переключите символьный набор командой GO SUB 8, теперь Вы будете видеть набираемый текст таким, каким он будет на экране. Обратное переключение символьного набора - GO SUB 9. Обратите внимание: в строках 1110 и 1210 в кавычках вводится символ "A", используя графический регистр. В строках 1370-1390 а также 1410-1460 в кавычках вводятся токены ключевых слов.

После того, как набор Бейсик-файла будет закончен, сделайте RUN 2 и загрузите заготовленные заранее кодовые файлы. После этого программа начнет работать. После того, как Вы убедитесь, что ошибок нет или устраните имеющиеся, можно убрать RETURN из строки 10. Запустите программу: RUN и посмотрите, как она будет вести себя при нажатии клавиши BREAK. Для остановки программы используйте "жучок" в строке 1365: как говорилось выше, если установить чёрный цвет чернил и черный цвет бумаги, то программу можно остановить, введя восклицательный знак (SYMB.SHIFT+1). Затем можно выгрузить полностью готовую программу на магнитную ленту, выполнив RUN 5.

Теперь пришло время несколько слов сказать о том, как адаптировать программу под "БЕТА-ДИСК интерфейс". Прежде всего, надо изменить строки 2 и 3 Бейсик-файла, добавив перед ключевым словом "LOAD" префикс:

```
RANDOMIZE USR 15619: REM :
```

Кроме того, надо будет изменить подпрограммы, связанные с загрузкой и выгрузкой экранов, начинающиеся со строк 70 и 80. БЕТА-вариант этих подпрограмм:

```
70 LET err=USR 15619 : REM : LOAD f$ CODE 16384
72 IF err<>0 THEN GO SUB 60
79 RETURN
80 RANDOMIZE USR 15619: REM : ERASE f$ CODE
82 LET err=USR 15619: REM : SAVE f$ CODE 16384,6912
84 IF err<>0 THEN GO SUB 60
89 RETURN
```

В подпрограмме записи файла сначала производится удаление предыдущей версии файла на диске с тем именем, которое было задано, а затем происходит запись экрана в новый файл с таким именем. В случае ошибки чтения-записи происходит вызов

подпрограммы GO SUB 60, которая выводит предупредительную табличку. Для простоты программы не предусмотрена верификация. Вы можете сами организовать ее, если захотите. Кроме того, если предварительное стирание файла с заданным именем Вас не устраивает, то просто исключите строку 80. В результате, в том случае, если на диске уже имеется файл с таким именем, то записи экрана на диск не произойдет, а будет выдано сообщение об ошибке подпрограммой GO SUB 60.

В строке 52 вместо "макс. 10 символов" подставьте: "макс. 8 символов". В строке 54 удалите все, начиная с IF … до конца строки.

На этом адаптацию под диск можно считать законченной.

А теперь дополнительная информация для тех, кто занят творческим поиском, а не просто копирует готовую программу.

Нам почему-то гораздо симпатичнее вариант, когда все блоки кодов, используемые в программах, находятся в нулевой строке, в области за оператором REM. В частности, тогда полностью исключается несанкционированная остановка программы, так как блок кодов ON ERROR GO TO, располагаясь внутри Бейсик программы, может быть инициирован сразу же после старта программы. Это обстоятельство, дополненное грамотно продуманными приемами защиты, создаст больше хлопот для взломщиков. Программа "МОЗАИКА", конечно, не претендует на такую серьезную защиту, но речь сейчас о том, какие методы Вы можете использовать в своих разработках, которые, возможно, как раз потребуют хорошей защиты. Еще одно обстоятельство в пользу кодов в нулевой строке. Такая программа гораздо быстрее загружается (это относится как к магнитофонному, так и к дисковому вариантам). А это уже существенное преимущество для пользователя. Все определяется только тем, "стоит ли овчинка выделки", то есть какими трудами это дается программисту. Попробуем показать, что в нашем случае эти дополнительные эти труды совсем незначительны.

При разработке "МОЗАИКИ" мы исходили из того, что программа должна быть легко адаптируема под "БЕТА-ДИСК интерфейс". О путях решения этой проблемы уже говорилось в "ZX-РЕВЮ" N 9-10 за 1992 г. стр. 197. Но там была ситуация, когда все блоки кодов в нулевой строке были перемещаемыми в памяти. Проблема была только в том, чтобы обеспечить правильное обращение к ним из Бейсика. В программе "МОЗАИКА" используется неперемещаемый блок кодов, воспроизводящий мелодию! Если даже его скомпилировать при помощи музыкального редактора "WHAM" для расположения в нулевой строке магнитофонного варианта, тогда при адаптации под "БЕТА-ДИСК" он окажется смещенным на 112 байт и не будет работать.

Для универсальности программы применяется следующий прием. После загрузки программы все рабочие блоки кодов из нулевой строки перебрасываются в отведенное для них место (с адреса 63824) и работают уже там. (Программа "МОЗАИКА" небольшая, с количеством свободной памяти проблем нет.)

В начале нулевой Бейсик-строки находится процедура, осуществляющая переброску кодового блока заданной длины, следующего непосредственно за этой процедурой, на заданный адрес. Этот способ Вы можете использовать и в других своих программах. Независимо от того, где располагается сама процедура, адрес, откуда будет произведена переброска, всегда берется следующим за концом процедуры. Вот она:

```
5CD0 CD7C00     CALL  #007C       (1)
5CD3 3B         DEC   SP          (2)
5CD4 3B         DEC   SP
5CD5 E1         POP   HL
5CD6 011000     LD    BC,#0010    (3)
5CD9 09         ADD   HL,BC
5CDA 1150F9     LD    DE,#F950    (4)
5CDD 019606     LD    BC,#0696
5CE0 EDB0       LDIR              (5)
5CE2 C9         RET
5CE3 .... перебрасываемый массив
```

Процедура имеет длину 19 байт и расположена в начале нулевой строки сразу же за REM, адрес в случае магнитофонного варианта равен 23760 (#5CD0). Но с одинаковым

успехом она будет работать и в любом другом месте. Кодовый массив, подлежащий переброске, располагается сразу же за процедурой. Для определения этого адреса используется такой прием. Сначала выполняется подпрограмма из ПЗУ (1). По адресу #007C в ПЗУ находится только одна команда RET. Но при выполнении инструкции CALL на стек записывается адрес команды, следующей за CALL, то есть в данном случае #5CD3. Для того, чтобы осуществить привязку к адресам, это значение возвращается (2) в регистр HL. Далее это значение увеличивается на величину смещения, заданного в регистре BC (3). Теперь в HL получилось: #5CD3 + #0010 = #5CE3. Затем задаются остальные параметры для выполнения команды LDIR (4). В DE - адрес места назначения перебрасываемого блока: 63324, а в BC - его длина: 1684 байта. (Конкретные значения подставлены те, которые используются в программе "МОЗАИКА".) Теперь выполняется команда LDIR (5) и происходит возврат в вызывающую программу.

В нулевой строке "МОЗАИКА" расположены следующие блоки кодов:

1. Процедура, осуществляющая переброску рабочих кодов в адрес 63824. Длина - 19 байт.

2. Рабочие коды (перебрасываемые в адрес 63824). Длина - 1684 байта.

Суммарная длина кодов, располагаемых в нулевой строке, равна: 19+1684=1703 байта.

При создании нулевой строки с большим объемом памяти после REM, Вы можете воспользоваться программой REM FILL, ранее опубликованной в "ZX-РЕВЮ" N 9-10 за 1992 г. на стр. 194-196. Сегодня же предложим новую версию этой программы. Она называется "REM 2". Кодовый блок этой версии имеет длину всего 62 байта. Это оказалось возможным осуществить благодаря использованию подпрограммы из ПЗУ, расположенной по адресу 5717, которая позволяет зарезервировать заданный объем памяти, раздвинуть текст Бейсик-строк и перерасчитать все системные переменные. Она используется процедурами редактора Спектрума при добавлении новых строк в программу. (О ней, в частности, упоминалось в "ZX-РЕВЮ" N 5-6 в статье Пашорина В.И. на стр. 112.) Программа "REM 2" состоит только из Бейсик-файла (кодовый блок формируется после старта Бейсик-программы):

```
1 REM
10 LET n=23296: LET s=0
20 FOR x=n TO n+61
30 READ y: POKE x,y: LET s=s+y
40 NEXT x
50 IF S<>6108 THEN PRINT  FLASH 1; "ERROR IN LINE DATA": STOP
60 INPUT "No of extra bytes: "; n
70 RANDOMIZE n
80 POKE 23312,PEEK 23670: POKE 23313,PEEK 23671
100 DATA 042,083,092,229,054,000,035,054,000,035,094,035,086,213,025,001,008,000,197,205,
      085,022,035,229,209,019,054
110 DATA 000,193,197,011,237,176,209,225,025,235,225,035,035,115,035,114
120 DATA 000,033,001,000,205,110,025,229,033,016,039,205,110,025,209,205,229,025,201
200 CLEAR : RANDOMIZE USR 23296
```

Подробно принцип работы этой программы был изложен в указанной выше статье в "РЕВЮ" N 9-10. Поэтому, чтобы не повторяться скажем только, что после старта программа запрашивает число дополнительных байтов, которые будут вставлены между последним символом текста начальной строки и символом <ENTER>, завершающим эту строку. После ввода этого параметра происходит старт кодового блока. После сообщения 0: OK, область за REM в 1-й строке будет расширена на число добавочных байт и обнулена, номер этой строки станет 0, а все остальные Бейсик-строки будут уничтожены. Теперь в полученную REM область можно загрузить коды.

Первое значение DATA в строке 110 определяет код заполнения дополнительной области. Если в строке 120 первое значение DATA заменить на 201, то уничтожения остальных Бейсик-строк не будет. Экспериментируя с указанными значениями, удалите строку 50, которая проверяет контрольную сумму значений DATA, защищая от ошибок при наборе программы.

Теперь возвращаемся к "МОЗАИКЕ". На вопрос о числе добавочных байтов ответьте:

1703. После того, как нулевая строка с областью заданной длины после REM будет сформирована, можете загружать в нее коды и объединить ее при помощи MERGE с Бейсик-файлом "МОЗАИКИ". Если Вы используете БЕТА-ДИСК, то на всякий случай загляните в статью о TR-DOS в этом выпуске "РЕВЮ". Там автор упомянул об особенностях загрузки кодов в нулевую строку в отличие от магнитофонного варианта.

Теперь осталось изменить строки 2 и 3 Бейсик-файла "МОЗАИКА":

```
2 BORDER 7: PAPER 7: INK 0: CLEAR 49999: POKE 23739,111: RANDOMIZE USR (PEEK 23635+256*PEEK
      23636+5)
3 RANDOMIZE 4: GO SUB 10: LOAD "mozaik $"CODE 16354
```

В заключение же хочется пожелать всем юным пользователям "Спектрума" приятной работы с программой "МОЗАИКА".

Стюарт Николс Перевод английского: Пашорин В. И.

ПРИМЕНЕНИЕ АССЕМБЛЕРА ДЛЯ СОЗДАНИЯ БЫСТРОРАБОТАЮЩИХ ПРОГРАММ

ПРЕДИСЛОВИЕ

Известные книги, справочники и руководства по машинному языку процессора Z-80 используемого в компьютере "ZX-Spectrum" в основном ограничиваются описанием действий всех команд процессора и порядком применения этих команд на простейших примерах программ в машинных кодах. В лучшем случае приводятся программы для сложения чисел или им подобные.

Практика показывает, что конкретному пользователю, желавшему начать освоение машинного кода нужен немного другой подход. Нужна книга, которая поможет ему немедленно, сразу взяться за дело и почувствовать мгновенную отдачу от своих усилий. Одним словом, нужна простейшая практика и живая заинтересованность.

Как нам кажется, в основу этой книги положена весьма плодотворная идея - опереться на знание пользователем БЕЙСИКа и постепенно развивать эти знания в направлении машинного кода, шаг за шагом показывая как алгоритмы БЕЙСИКа переносятся на язык АССЕМБЛЕРа. Эта концепция очень хорошо укладывается в тот стиль работы, который "ИНФОРКОМ" ведет уже третий год на страницах "ZX-РЕВЮ" и в прочих своих книгах.

Нам также показалась очень привлекательной идея книги дать широкое представление конкретному программисту о возможности использования процедур, "зашитых" в ПЗУ для своей повседневной работы. Это может заинтересовать и тех, кто достаточно хорошо знает "АССЕМБЛЕР, но не имеет доступной информации об использовании процедур ПЗУ в своих программах.

Мы горячо благодарим нашего постоянного соавтора из г. Балашова Саратовской обл. В. Пашорина за прекрасный перевод этой нужной книги и, главное, за достойный выбор самой книги для перевода.

К сожалению, мы не могли бы сказать, что оригинал написан простым доступным языком, понятным каждому неспециалисту. Но мы очень хорошо потрудились над ее адаптацией. Оставив по сути без изменения структуру книги и ее фактическое содержание, мы полностью переработали всю методику подачи материала и надеемся, что сделали ее доступной для неподготовленного читателя, впрочем о том, как это удалось, судить будете Вы сами.

Поскольку сама книга фактически не является учебником по машинному кодированию, а служит практическим пособием для тех, кто пошел по этому пути, мы должны напомнить Вам о целесообразности приобретения нашего издания "Программирование в машинных кодах. Первые шаги, практикум. Справочник". По этой книге уже обучились тысячи пользователей и она пока продолжает оставаться лучшим из всего, что было написано на данную тему.

Для тех наших читателей, которые сейчас решают вполне естественный вопрос о целесообразности приобретения всего комплекта "ZX-РЕВЮ-93", мы даем содержание книги, которая войдет в "РЕВЮ-93" полностью:

1. Вывод на экран.
2. Команды PLOT, DRAW, CIRCLE.
3. Счет очков в программах.
4. Случайные числа.
5. Опрос клавиатуры.
6. Перемещение объектов.
7. Музыкальные и звуковые эффекты.
8. Команды ATTR, SCREEN$, POINT.

9. Работа с принтером.

10. Разбор конкретного примера перевода игровой программы из БЕЙСИКа в машинный код.

1. ВЫВОД НА ЭКРАН

Создание программ в машинных кодах, выполняющих вывод информации на экран - утомительная задача, требующая значительных затрат рабочего времени. Тем не менее, это одна из наиболее важных задач. Даже очень хорошие программы проиграют многое, если информация, выводимая на экран при их выполнении, не вполне понятна или, хуже того, допускает неоднозначную трактовку. Перед созданием любой программы необходимо подумать о том, чтобы наглядность выводимой на экран информации была обеспечена на должном уровне.

На компьютере "ZX_SPECTRUM" можно создать 21 графический элемент, форма которых полностью определяется пользователем. Это так называемые символы графики пользователя (User Definable Graphics - UDG). Необходимо как можно полнее использовать эти символы в своих программах при выводе информации на экран. Например, при создании игровой программы типа "космических завоевателей" для получения отдельных фрагментов изображения экрана можно использовать те же символы, что используются для создания самих "завоевателей" или других объектов.

Для вывода информации на экран целесообразно использовать контрастные цвета. Не рекомендуется использовать вместе красный и розовый, желтый и белый и т.п. Благоразумно в программах использовать только черный и белый цвета, так как не все пользователи имеют цветные мониторы или цветные телевизоры.

Так как выводить информацию на экран необходимо при работе практически любой программы, то мы начнем нашу книгу с того, что рассмотрим в этой главе способы перевода команды БЕЙСИКа PRINT в машинный код.

Обычно БЕЙСИК-программа начинается с того, что определяются основные цветовые атрибуты, например:

```
10 PAPER 3: INK 1: BORDER 5
```

Затем эти атрибуты устанавливаются в системных переменных компьютера и становятся действующими. Делают это командой CLS.

```
20 CLS
```

Давайте попробуем сделать то же самое в машинном коде. Самой простой будет операция установления цвета бордюра. Она займет всего лишь 6 байтов в оперативной памяти.

Сначала в регистр A процессора заносится число 5 (оно означает зеленый цвет бордюра), а затем вызывается процедура ПЗУ, которая выполнит изменение этого цвета. Данная процедура называется BORDER и находится она в ПЗУ по адресу 229BH (8859 DEC).

Процедура не только установит необходимый цвет окантовки экрана, но и передаст значение номера цвета в системную переменную BORDER (23624 DEC - 5C48H). Для хранения номера цвета бордюра в этой системной переменной отведены 3 бита (с третьего по пятый) и, таким образом, всего могут быть установлены до 8 различных цветов. Оставшиеся биты в этой системной переменной используются для хранения цветовых атрибутов нижней части экрана (системного окна). Системное окно занимает как правило (не всегда) две нижние строки экрана, в которых Вы выполняете INPUT или редактирование программы. Биты 0...3 определяют номер цвета INK в системном окне, бит 6 отвечает за параметр BRIGHT, а бит 7 - за параметр FLASH.

Итак, программа в машинных кодах, эквивалентная команде БЕЙСИКа BORDER 5, будет выглядеть так, как показано на листинге 1.1 (Предполагается, что машинной код начинается с адреса, заданного директивой АССЕМБЛЕРа ORG).

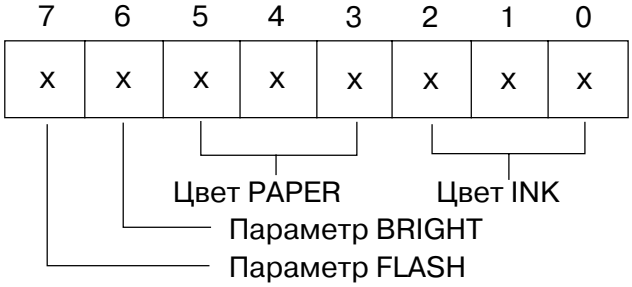
Листинг 1.1

```
АДРЕС  МАШ.КОД     АССЕМБЛЕР    КОММЕНТАРИЙ
                   ORG 23760
23760  3E 05       LD A,5       ;Установили в аккумуляторе число
                                ;"5" - код зеленого цвета.
```

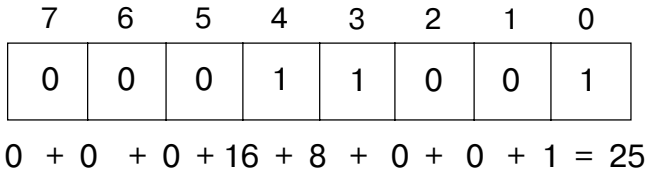
```
23762 CD 9B 22    CALL 8859     ;Вызов процедуры ПЗУ "BORDER".
23765 C9          RET           ;Выход.
```

Выполнить установку цветов INK и PAPER немного сложнее и соответствующая программа в машинных кодах у нас займет уже не 6, а 13 байтов (она приведена на листинге 1.2).

За установку цветов PAPER и INK в основной части экрана отвечает системная переменная ATTR-P (23693 DEC = 5C8D H). Поэтому установка этих цветов из машинного кода сводится к переключению битов в данной системной переменной, а раскладка этих битов такова:



Если мы хотим из машинного кода дать команду, аналогичную PAPER 3; INK 1, то нам надо включить биты в системной переменной ATTR_P следующим образом:



Т.е. фактически нам надо было бы в ячейку памяти с адресом 23693 заслать число 25, например так:

```
LD (23693),25
```

Но такой команды, которая позволила бы заслать произвольное число в произвольную ячейку памяти компьютера у процессора Z-80 нет. Приходится немного комбинировать.

Сейчас у нас есть возможность познакомиться с интересным способом адресации в процессоре Z-80, который называется индексной адресацией. Дело в том, что в процессоре есть шестнадцатиразрядная регистровая пара IY, в которой может храниться какое-либо целое число от 0 до 65535. Работа компьютера "ZX-Spectrum" организована таким образом, что после включения его в сеть в этой регистровой паре устанавливается число 23610 (Это адрес системной переменной ERR_NR). И этим активно пользуются на практике. Есть неофициальная договоренность, что для работы с системными переменными используется эта регистровая пара. При этом не надо указывать адрес нужной Вам системной переменной, а достаточно указать "смещение", т.е. величину, на которую отстоит нужный Вам адрес от заранее установленного в регистре IY. Тогда вместо LD (23693),25 можно записать команду:

```
LD (IY+83),25
```

Эта запись выглядит понятнее уже хотя бы потому, что все, кто программируют на "Спектруме" в машинных кодах, сразу мгновенно понимают: раз используется IY, значит здесь загружают какую-то системную переменную. А какую именно, они быстро определят, ОТКРЫВ таблицу системных переменных.

Вернемся к нашему примеру, см. листинг 1.2. Итак, в первой строчке мы установили необходимые цвета PAPER и INK.

Листинг 1.2 .

```
АДРЕС  МАШ.КОД          АССЕМБЛЕР          КОММЕНТАРИЙ
                        ORG 23760
23760  FD 36 53 19      LD (IY+83),25      ;Установили в системной перемен-
                                           ;ной BORDER значения INK и PAPER.
23764  3E 02            LD A,02            ;Подготовка к открытию канала 2.
23766  CD 01 16         CALL 5633          ;Открываем канал 2.
23769  CD 6B 0D         CALL 3435          ;Вызов процедуры CLS.
23772  C9               RET                ;Выход.
```

Листинг 1.3

```
АДРЕС  МАШ.КОД          АССЕМБЛЕР          КОММЕНТАРИЙ
                        ORG 23760
23760  3E 02            LD A,2             ;Подготовка к открыванию канала.
23762  CD 01 16         CALL 5633          ;Открываем канал экрана.
23765  3E 41            LD A,65            ;Код символа "A".
23767  D7               RST 16             ;Вызов процедуры печати символа.
23768  C9               RET                ;Выход.
```

Листинг 1.4

```
АДРЕС  МАШ.КОД          АССЕМБЛЕР          КОММЕНТАРИЙ
                        ORG 23760
23760  3E 02            LD A, 2            ;Подготовка к открыванию канала.
23762  CD 01 16         CALL 5633          ;Открываем канал экрана.
23765  3E 48            LD A,72            ;Код символа "H".
23767  D7               RST 16             ;Вызов процедуры печати символа.
23768  3E 45            LD A,69            ;Код символа "E".
23770  D7               RST 16             ;Вызов процедура печати символа.
23768  C9               RET                ;Выход.
```

Теперь мы должны дать команду CLS. Это тоже сделает за нас процедура, записанная в машинном коде ПЗУ. Данная процедура так и называется - CLS и находится по адресу 0D6BH = 3435 DEC.

Вместе с тем, Вы по-видимому обратили внимание на то, что перед вызовом этой процедуры вызывается еще одна процедура ПЗУ по адресу 5633 (1601H). Это весьма важная процедура, она называется CHAN_OPEN, и мы будем еще не раз использовать ее при изучении материалов данной главы. Вызовом этой процедуры открывается канал вывода данных на экран - это канал под номером 2 и потому перед вызовом процедуры в регистр A загружается число 2. Когда Вы работали в БЕЙСИКе, то могли не думать о назначении каналов. Если же Вы выдаете информацию из машинного кода, то никто за Вас этого не сделает, а сделать это необходимо, иначе компьютер не поймет, куда надо подавать информацию.

Теперь Вы, уважаемые читатели, должны уметь устанавливать в машинном коде атрибуты экрана и выполнять очистку экрана. Давайте посмотрим как же нам напечатать на экране что-нибудь содержательное. Проще всего напечатать на экране какой-либо символ. В листинге 1.3 показан пример программы, которая напечатает символ "A" (символ имеет номер 65 DEC) в левом верхнем углу экрана с координатами позиции печати (0,0). Именно такие координаты текущей позиции устанавливаются после того, как отработает процедура CLS.

Если Вы хотите распечатать на экране какой-либо символ, то его код предварительно должен быть заслан в регистр A процессора, после чего должна быть выдана команда RST 16, что Вы и видите на листинге 1.3. По команде RST 16 процессор запустит процедуру ПЗУ PRINT_A_2, находящуюся по адресу 15F2H = 5618 DEC. Символ, код которого находится в аккумуляторе будет напечатан и осуществится переход к следующей позиции печати.

Если с этим Вам все понятно, то Вы без труда поймете работу программы 1.4, которая напечатает целое слово - "HE", поочередно засылая символы в аккумулятор и выдавая их командой RST 16.

Хорошо, что слово "HE" имеет только два символа, а как быть, если нужно напечатать

длинное предложение? Понятно ведь, что кодировать печать длинной фразы по символам дело очень утомительное, к тому же необходим большой расход оперативной памяти - он в два раза больше, чем длина Вашего сообщения, поскольку на каждый символ еще нужна команда RST 16, занимающая байт.

Для решения этой проблемы существуют как минимум три пути.

Главная идея состоит в том, чтобы записать все символы выводимого сообщения в некоторый банк данных и считывать их оттуда по одному до тех пор, пока весь банк не будет исчерпан. Этот прием показан в Листинге 1.5 (версия 1).

В этом листинге Вы знакомитесь с двумя новыми регистровыми парами - BC и DE. Как правило, пара BC используется программистами для организации в ней разного рода счетчиков, а пара DE - для указания на какой-либо адрес в памяти компьютера. Регистры B,C,D и E не обязательно должны использоваться парами, как это было с регистром IY. Они могут использоваться и как одиночные восьмибитные регистры для хранения целых чисел от 0 до 255.

Обратите внимание на пару операций

```
LD A,B
OR C
```

Это очень быстрый, экономичный и к тому же общепринятый способ проверки BC на ноль. Поскольку в регистре A у нас загружено содержимое регистра B, то результат операции OR C будет нулевым в том и только в той случае, если и B и C равны нулю.

Листинг 1.5 (Версия 1).

```
АДРЕС  МАШ.КОД      АССЕМБЛЕР          КОММЕНТАРИЙ
                    ORG 23760
23760  3E 02        LD A,2             ;Подготовка к открыванию канала.
23762  CD 01 16     CALL 5633          ;открываем канал экрана.
23765  11 E4 5C     LD DE,DATA         ;В регистровую пару DE загрузили
                                       ;адрес, в котором хранится наше
                                       ;текстовое сообщение.
23768  01 07 00     LD BC,7            ;В регистровой паре BC организо-
                                       ;вали счетчик на 7 символов.
LOOP
23771  78           LD A,B             ;Проверка не обнулился ли счет-
23772  B1           OR C               ;чик символов в BC?
23773  C8           RET Z              ;Выход, если он обнулялся.
23774  1A           LD A,(DE)          ;Взяли в аккумулятор текущий
                                       ;символ из списка DATA.
23775  D7           RST 16             ;Напечатали его на экране.
23776  13           INC DE             ;Перешли к очередному символу.
23777  08           DEC BC             ;Уменьшили счетчик символов на 1.
23778  18 F7        JR LOOP            ;Перешли на метку LOOP для оче-
                                       ;редной проверки не закончился
                                       ;ли список печатаемых символов.
DATA
23779  72 69 76 80 32 77 69           ;Начиная с адреса 23779 мы храним
                                       ;сообщение "HELP ME".
```

Листинг 1.5 (Версия 2).

```
АДРЕС  МАШ.КОД      АССЕМБЛЕР          КОММЕНТАРИЙ
                    ORG 23760
23760  3E 02        LD A,2             ;Подготовка к открыванию канала.
23762  CD 01 16     CALL 5633          ;Открываем канал экрана.
23765  11 DF 5C     LD DE,DATA         ;Адрес сообщения.
23768  01 07 00     LD BC,7            ;Длина сообщения - 7 символов.
23771  CD 3C 20     CALL 8252          ;Вызов процедуры ПЗУ PR_STRING.
23774  C9           RET                ;Выход.
DATA
23775  72 69 76 80 32 77 69           ;Сообщение "HELP ME".
```

Обратите также внимание на то, что мы записали наше сообщение, начиная с адреса 23779. Но это совсем не обязательно. Можете хранить свои тексты где хотите, в любых адресах, удобных Вам. Важно только, чтобы в строке LD DE,DATA в регистровую пару DE был заслан истинный адрес начала Вашего сообщения ОТ 0 ДО 65535.

Второй вариант этой программы выглядит немного более простым. Если Вы заранее установили в регистровой паре DE адрес, с которого начинается Ваше сообщение, а в BC - длину этого сообщения, то больше можно вообще ничего не делать, а вызвать процедуру ПЗУ под названием PR_STRING, которая сама напечатает за Вас это длинное сообщение. Процедура PR_STRING находится по адресу 8252 DEC = 203CH. Пример см. в Листинге 1.5 (версия 2).

Третий способ - самый мощный. Его применяют в тех случаях, когда у Вас заготовлено не одно сообщение, а много, т.е. Вы имеете целую таблицу сообщений, но заранее не знаете, какое когда придется печатать. В этом случае Вам хотелось бы выдавать на экран нужное сообщение указав только его номер. Именно так и поступают в абсолютном большинстве программ, написанных в машинных кодах. Именно так организовано и само ПЗУ, которое, как Вы знаете, хранит в себе много разных сообщений типа "Start tape and press key..." и выдает вам всякий раз именно то, которое надо.

Чтобы программа знала, где кончается одно сообщение и начинается другое, применяют специальный прием - к коду последнего символа сообщения прибавляют число 127, т.е. фактически в последнем символе каждого сообщения принудительно включают старший (7-ой) бит. Таким образом, символ, код которого > 127 является маркером конца сообщения. Теперь нетрудно создать программу для печати 1-го сообщения. Она просмотрит тексты, начиная с адреса, установленного в DE, отсчитает N-1 маркер и начнет печатать то, что найдет после него. Символы будут печататься по одному до тех пор, пока не будет найден N-ый маркер. Для печати последнего придется предварительно отнять от его значения число 127.

В листинге 1.5 приведен пример такой работы, но для простоты там принято, что у Вас есть не таблица, а только одно сообщение, заканчивающееся инвертированным символом (к коду символа прибавлено число 127).

Кстати, обратите внимание на то, что если числа, большие чем 27, являются маркерами конца сообщения, то очевидно нельзя допускать, чтобы в Ваших текстах были символы, код которых больше 27, т.е. символы блочной графики коды от 128 до 143) и символы графики пользователя (коды от 144 до 164) воспроизводить таким методом нельзя.

Похожая на эту, но более сложная процедура вывода на экран значений клавишных слов и сообщений об ошибках используется в ПЗУ компьютера. Стартовый адрес этой процедуры - 3082 -DEC = 0C0AH.

Вы можете и сами выдавать из машинного кода системные сообщения об ошибках, полный перечень которых обычно приводят в руководстве пользователя компьютера.

Так, они могут быть легко выведены на экран с помощью команды RST 8. Байт, следующий за этой командой, определяет какой вид сообщения необходимо вывести, например, если необходимо вывести на экран сообщение "K invalid colour", которое в перечне сообщений об ошибках является двадцать первым по счету, то необходимо записать:

```
RST 8
DEFB 19 (номер сообщения минус 2)
```

Причем нет необходимости давать команду RET после этого, так как сразу же после вывода на экран сообщения об ошибке осуществляется возврат в BASIC-систему.

Теперь мы можем приступить к рассмотрению способов вывода на экран символов или даже целых строк в заданную позицию на экране. В BASIC-программах это выполняется с помощью оператора PRINT AT, после которого записывается номер строки, номер столбца и, наконец, само сообщение, которое необходимо вывести на экран.

Для этого используются так называемые управляющие коды. Если Вы посмотрите любую таблицу символов ASCII, то увидите, что печатным символам соответствуют коды, начиная с 32 DEC (это символ "пробел". А коды с 0 до 31 является непечатными (т.е. не имеют закрепленной за ними литеры) и используются в качестве управляющих. Так,

например, в "Спектруме" коды с 0 по 5 и с 24 по 31 вообще не используются, а с 6 по 23 - служат в качестве управляющих. (В компьютерах других систем это может быть и не так).

За координаты позиции печати отвечает код 22, он является аналогом БЕЙСИКовского оператора AT. К нашему счастью, команда RST 16 распознает код 22 как PRINT AT и использует следующие 2 байта как "x" и "y" координаты, задающие PRINT-позицию для вывода сообщения на экран. Программа 1.6 демонстрирует использование последовательности байтов

```
DEFB 22
DEFB 3
DEFB 5
```

как аналог оператора

```
PRINT AT 3,5.
```

Совершенно так же можно применять коды 23 и 13 как аналоги операторов TAB и ENTER, соответственно.

Листинг 1.5 (Версия 3).

```
АДРЕС  МАШ.КОД      АССЕМБЛЕР          КОММЕНТАРИЙ
                    ORG 23760
23760  3E 02        LD A,2             ;Подготовка к открыванию канала.
23762  CD 01 16     CALL 5633          ;Открываем канал экрана.
23765  11 E5 5C     LD DE,DATA         ;Адрес начала сообщения.
LOOP
23768  1A           LD A,(DE)          ;Взяли текущий символ.
23769  CB 7F        BIT 7,A            ;Проверяем его старший бит.
23771  20 04        JR NZ,END          ;Если он не нулевой, то это пос-
                                       ;ледний символ и мы переходим на
                                       ;метку END.
23773  D7           RST 16             ;Иначе печатаем его на экране.
23774  13           INC DE             ;Перешли к очередному символу.
23775  18 F7        JR LOOP            ;Возврат на метку LOOP.
END
23777  CB BF        RES 7,A            ;Выключили старший бит последнего
                                       ;символа.
23779  D7           RST 16             ;Напечатали последний символ.
23780  C9           RET                ;Выход DATA
23781  72 69 76 80 32 77 197          ;Сообщение "HELP ME".
```

Листинг 1.6

```
АДРЕС  МАШ.КОД      АССЕМБЛЕР          КОММЕНТАРИЙ
                    ORG 23760
23760  3E 02        LD A,2             ;Подготовка к открыванию канала.
23762  CD 01 16     CALL 5633          ;Открываем канал экрана.
23765  11 DF 5C     LD DE,DATA         ;Адрес начала сообщения.
23768  01 0C 00     LD BC,12           ;Длина сообщения с учетом управляющих кодов.
23771  CD 3C 20     CALL 8252          ;Вызов процедуры ПЗУ PR_STRING
23774  C9 RET                          ;Выход DATA
23775  22           DEFB 22            ;Код управления позицией печати
23776  03           DEFB 3             ;Координата "Y=3"
23777  05           DEFB 5             ;Координата "X=5"
23778  84 72 65 78 4B                  ;Текст сообщения "THANK YOU"
23783  32 89 79 65
```

Листинг 1.7

```
АДРЕС  МАШ.КОД      АССЕМБЛЕР          КОММЕНТАРИЙ
                    ORG 23760
23760  3E 02        LD A,2             ;Подготовка к открыванию канала.
23762  CD 01 16     CALL 5633          ;Открываем канал экрана.
23765  11 E2 5C     LD DE,DATA         ;Адрес начала сообщения.
23768  01 24 00     LD BC,36           ;Длина сообщения с учетом управляющих кодов.
23771  CD 3C 20     CALL 6252          ;Вызов процедуры ПЗУ PR_STRING
23774  CD 4D 0D     CALL 3405          ;Копирование ATTR-P в ATTR-T.
23777  C9           RET                ;Выход DATA
23778  22 04 07 17 02                  ;Сообщение CAN YOU READ THIS c
```

```
33783  16 04 18 01 19                ;инплантированными в нем кодами
23788  01 21 01 20 01                ;управления координатой позиции
23793  67 65 78 32 89                ;печати и кодами управления
23798  79 85 32 82 69                ;цветовыми атрибутами.
23803  65 68 32 84 72
23808  73 83 21 00 20
23813  00
```

С помощью управляющих кодов можно не только задавать координаты позиции печати, но и цвета и прочие атрибуты печатаемых символов. Для этого есть свои управляющие коды.

INK - код 16
PAPER - код 17
FLASH - код 18
BRIGHT - 19
INVERSE - 20
OVER - код 21

Программа 1.7 показывает как используются управляющие коды, соответствующие PAPER, INK, BRIGHT и т.д.

После использования временных цветов для вывода на экран сообщения необходимо вернуться к постоянным цветам для обеспечения дальнейшего вывода. Для режимов OVER и INVERSE это требование легко выполняется, так как команда RST 16 легко распознает их управляющие коды. Для изменения PAPER, INK, BRIGHT и FLASH, необходимо вызвать из ПЗУ процедуру (адрес 3405), которая копирует содержимое системной переменной ATTR-P (23693) в системную переменную ATTR-T(23695).

(Продолжение следует)

Михайленко В.С., Турович А.К.

ЗАЩИТА ПРОГРАММ

Мы продолжаем публикацию материалов, посвященных защите программного обеспечения от просмотра и копирования. Непосредственная тема сегодняшней статьи связана с подробным разбором стандартных процедур ПЗУ, осуществляющих загрузку данных с магнитофона.

(ПРОДОЛЖЕНИЕ)

Начало см. в "ZX-РЕВЮ-92":
N 1,2 - с. 9-16
N 3,4 - с. 53-60
N 5,6 - с. 97-104
N 7,8 - с. 141-146
N 9,10 - с. 185-192
N 11,12 - с. 230-237.

1.1 Процедуры в машинных кодах, реализующие загрузку программ с магнитофона.

Мы с Вами рассмотрели систему кодирования информации на магнитной ленте. Теперь приступим к рассмотрению процедур ПЗУ, реализующих ввод программ "Спектрума" с магнитофона.

Для того, чтобы точно оценить временные характеристики операций, реализующих необходимые задержки в цикле загрузки, необходимо знать время исполнения отдельных инструкций процессора Z-80. Как Вам уже вероятно известно, скорость работы процессора на несколько порядков выше максимальной скорости записи кодированных логических единиц информации на магнитной ленте. Поэтому программа должна выполнять задержку после каждого записываемого на магнитофон бита данных.

Время задержки удобно считать не в секундах, а в циклах тактового генератора, который синхронизирует работу процессора Z-80. У каждой команды процессора есть своя продолжительность, которая может быть измерена в этих циклах. Впрочем, не для всех команд это величина постоянная. Так, например, для команд, связанных с каким-либо условием, (например JR NZ и т.п.) число циклов обычно зависит от того, выполняется или нет данное условие и для каждого случая различно, поэтому нам необходимо учитывать и этот факт и рассматривать соответствующие приемы задержки, используемые в данной процедуре обработки информации, поступающие с магнитной ленты.

Информацию о количестве циклов по каждой команде процессора Вы можете получить из справочника "ИНФОРКОМа" по микропроцессору Z-80, см. книгу "Программирование в машинных кодах. Первые шаги. Практикум. Справочник".

Процедура, выполняющая загрузку информации с магнитофона, начинается в ПЗУ с адреса 0556H = 1366 DEC. Ее действие заключается в многократной проверке шестого бита порта 254 и в определении временного интервала между двумя следующими друг за другом фронтами.

Таким образом, при чтении блока данник с ленты важнейшим является точное определение интервала между фронтами сигнала. Эту работу выполняет специализированная процедура, размещенная в ПЗУ компьютера под адресом 05E3H = 1507. Эту точку входа в ПЗУ можно использовать, когда надо установить, сколько времени прошло от момента вызова процедуры до обнаружения двух, следующих один за другим, смен уровней сигналов в гнезде EAR (и насколько оба они соответствуют допустимым пределам требуемых временных интервалов).

Второй важной точкой входа в ПЗУ является специализированная процедура,

позволяющая определить, сколько времени прошло с момента вызова процедуры до получения первого фронта. Она размещается по адресу 05E7H = 1511. Вызов этой процедуры выполняется после помещения в регистр B временной константы, ограничивающей количество проверок порта 254.

Если оказывается, что время ожидания смены фронтов сигнала превышено или будет установлен факт нажатия клавиши BREAK, происходит сброс флага C флагового регистра F. Для распознавания каждой из этих двух ситуаций служит флаг Z. Если он обнулен, значит во время работы данной процедуры была нажата клавиша BREAK.

Мы начнем рассмотрение процедуры загрузки с изучения этих подпрограмм. Названия процедур стали общепринятыми после выхода в свет книги Я.Логана и Ф.О'Хары "Полный дисассемблер ПЗУ Спектрума". Те же названия использованы и в наиболее полном описании работы "Спектрума" с магнитофоном Анджея Каллофа ("Спектрум и магнитофон") и мы сохраняем те же наименования процедур.

```
        МЕТКА       АССЕМБЛЕР          ЦИКЛЫ
1       LD-EDGE-2   CALL LD-EDGE 1     ;17
2                   RET NC             ;11,5
3       LD-EDGE-1   LD A,#16           ;7
4       LD-DELAY    DEC A              ;1
5                   JR NZ,LD-DELAY     ;12,7
6                   AND A              ;4
```

Как Вы уже заметили, LD-EDGE-2 обеспечивает двукратный вызов процедуры LD-EDGE-1. Перед тем, как приступить к проверке порта, выжидается 358 тактов работы Z-80. процессор слишком быстродействующий по сравнению со временем, в течение которого на основе активного сигнала от ULA, стабилизируется состояние шестого бита порта 254.

В последней колонке приведено время затрачиваемое на выполнение каждой операции, измеренное в тактах микропроцессора. Так, в случае выполнения операции возврата из подпрограммы, в зависимости от состояния флага переноса C - RET NC - затрачиваются 5 или 11 тактов микропроцессора, в связи с тем, выполняется условие, или нет. По аналогичной причине мы приводим два значения и для процедуры условного перехода JR NZ, поскольку время, затрачиваемое на ее выполнение, может быть различным в зависимости от выполнения или невыполнения условия.

После начала работы процедуры LD-EDGE-1, в аккумулятор загружается шестнадцатеричное значение 16H. Благодаря этому мы организуем цикл задержки, основанный на уменьшении аккумулятора на единицу и сравнении этого значения с нулем. Если ноль не достигнут, мы продолжаем уменьшать аккумулятор на единицу, обеспечивая задержку. Так происходит ровно 7 + 21 X 16 + 4 + 7 + 4 = 358 тактов микропроцессора.

Командой AND A сбрасывается флаг C, который, как мы знаем, выполняет функции маркера.

После обнуления маркера проверяется, достигла ли временная константа в регистре B значения нуля. Обратите внимание, что в противоположность процедурам записи, здесь значение регистра B увеличено на единицу.

```
7       LD-SAMPLE   INC B              ;4
8                   RET Z              ;11,7
```

Следующим этапом идет контроль нажатия клавиши BREAK. Для этого мы считываем числовое значение из порта 254 и, после ротации данного значения (причем интересующий нас бит смещается во флаг переноса C), мы осуществляем продолжение работы, либо выходим из данной процедуры. Все зависит от того, была ли нажата клавиша BREAK.

```
9                   LD A,#7F           ;7
10                  IN A,(#FE)         ;11
11                  RRA                ;4
12                  RET NC             ;11,7
```

Теперь мы проверяем, изменился ли шестой бит 254 порта, в соответствии со значением, сохраняемым в регистре C (именно этот бит сигнализирует о поступлении сигнала с магнитофона). Если бит не изменился, о чем свидетельствует нулевой флаг Z, то он проверяется снова.

```
13              XOR C             ;4
14              AND #20           ;7
15              JR Z,LD-SAMPLE    ;32,7
```

Фронт сигнала извлечен, инвертируются все биты в регистре C и одновременно изменяется информация о состоянии порта, а также цвет изменяется на дополнительный цвет бордюра (красный-голубой или синий-желтый). Эти операции выполняются с помощью специальной команды инверсии. Для этого мы заносим в аккумулятор содержимое регистра C, после чего над аккумулятором выполняется команда CPL, которая инвертирует каждый его бит на противоположный. В результате получается как бы дополнение содержимого аккумулятора до 255 (в абсолютной двоичной арифметике).

После этого инвертированное значение мы возвращаем в регистр C.

```
16              LD A,C            ;4
17              CPL               ;4
18              LD C,A            ;4
```

После смены цвета бордюра на экране включается флаг C регистра F, что сигнализирует об успешном результате проверки. Соответствующие промежутки времени подсчитывает процедура, вызываемая на основе приращения значения в регистре B. Каждый вызов процедуры LD-EDGE-1 длится 465 тактов микропроцессора плюс по 58 тактов на каждый дополнительный тест.

```
19              AND #07           ;7
20              OR #08            ;7
21              OUT (#FE),A       ;11
22              SCF               ;4
23              RET               ;10
```

Рассмотрим теперь основную загрузочную процедуру LD-BYTES. Она занимает в памяти объем начиная от 0556H до 05E2H. Перед началом работы процедуры в регистр DE необходимо занести длину читаемого блока, в регистр IX адрес байта, начиная с которого должна быть загружена информация. В аккумулятор заносится тип блока.

LD-BYTES может не только считывать файлы в память, но также и сравнивать их. На этом принципе основано действие БЕЙСИК-команды VERIFY. Необходимый режим работы определяется состоянием флага переноса C перед вызовом процедуры загрузки. Его установка в единицу означает считывание в память всего файла, а обнуление - сравнение. Поскольку микропроцессор может работать не более чем с шестнадцатиразрядными числами, длина читаемого блока не может превышать 65535 (FFFFH).

```
1     LD-BYTES  INC D             ;4
2               EX AF,A'F'        ;4
3               DEC D             ;4
4               DI                ;4
5               LD A,#0F          ;7
6               OUT (#FE),A       ;11
7               LD HL,#053F       ;10
8               PUSH HL
```

Регистр D увеличивается с целью обнуления флага Z перед сохранением его в альтернативном наборе регистров микропроцессора. После этого значение регистра D

восстанавливается и запрещаются маскированные прерывания.

Занесение в регистр A значения CFH знаменует собой установку белого цвета бордюра путем выдачи этого значения в порт 254. В регистровую пару HL заносится адрес процедуры обработки возврата SA-LD-RET, который сохраняется на стеке.

```
 9                IN A,(#FE)         ;11
 10               RRA                ;4
 11               AND #20            ;7
 12               OR #02             ;7
 13               LD C,A             ;4
 14               CP A               ;4
```

Первый тест порта 254 представляет собой занесение в аккумулятор значения состояния данного порта с последующей его ротацией и выполнением операции логического "ИЛИ". Из всего байта оставляется только шестой бит, который характеризует состояние гнезда EAR. После выполнения этого цвет бордюра становится красным. После помещения в регистр C значения 22H или 02H включается флаг Z флагового регистра.

```
 15     LD-BREAK    RET NZ             ;11,7
 16     LD-START    CALL LD-EDGE-1     ;17
 17                 JR NC,LD-BREAK     ;12,7
```

Следующий отрезок программы представляет собой цикл, в котором программа будет задержана до нажатия клавиши BREAK или до смены состояния шестого бита порта 254. Во втором случае цвет бордюра будет изменен на голубой. Выполнение данной процедуры осуществляется многократным повторением вызова подпрограммы LD-BREAK.

```
 18                 LD HL,#0415        ;10
 19     LD-WAIT     DJNZ LD-WAIT       ;10,6
 20                 DEC HL             ;6
 21                 LD A,H             ;4
 22                 OR L               ;4
 23                 JR NZ,LD-WAIT      ;12,7
```

Перед началом непосредственного чтения информации с магнитной ленты выжидается около одной секунды. Это достигается за счет создания специального цикла задержки. Длительность ожидания зависит от значения, которое занесено в регистр HL. В нашем случае оно составляет #0415.

```
 24                 CALL LD-EDGE-2     ;17
 25                 JR NC,LD-BREAK     ;12,7
```

Первым делом проверяется, не передается ли еще какой-нибудь сигнал за данный промежуток времени. За время, составляющее приблизительно 14000 тактов микропроцессора должны появиться два фронта сигналов. Обратим внимание на большой излишек времени перед началом загрузки данных. В пилотирующем сигнале фронты появляются в интервале порядка 2186 тактов.

```
 26     LD-LEADER   LD B,#9C           ;7
 27                 CALL LD-EDGE-2     ;17
 28                 JR NC,LD-BREAK     ;12,7
 29                 LD A,#C6           ;7
 30                 CP B               ;4
 31                 JR NC,LD-START     ;12,7
```

В регистр B заносится временная константа 9CH, позволяющая процедуре LD-EDGE-2 искать два фронта за время, не превышающее 7000 тактов микропроцессора. С другой стороны, после их отыскания проверяется прошло ли с момента вызова вышеуказанной

процедуры до появления заднего фронта не менее, чем 3366 тактов работы микропроцессора. Таким способом отличаются фронты пилотирующего сигнала от значительно более близких пар фронтов информационных сигналов.

```
32                INC H              ;4
33                JR NZ,LD-LEADER    ;12,7
```

В цикле мы начинаем искать фронты пилотирующего сигнала. После извлечения 256 пар, мы переходим к ожиданию импульса синхронизации, который сигнализирует нам о начале собственно самого файла данных.

```
34     LD-SYNC    LD B,#C9           ;7
35                CALL LD-EDGE-1     ;17
36                JR NC,LD-BREAK     ;12,7
```

Проверяется время появления каждого нового фронта сигнала. Он должен быть обнаружен в течение 3655 тактов работы микропроцессора. Необходимая точность обеспечивается занесением в регистр B временной константы C9H и вызовом процедуры LD-EDGE-1.

```
37                LD A,B             ;4
38                CP #D4             ;7
39                JR NC,LD-SYNC      ;12,7
```

Однако, если это произошло до истечения 1100 тактов работы процессора, значит, уже появился первый фронт синхроимпульса. Для определения его появления мы заносим в аккумулятор содержимое регистра B и сравниваем полученное значение с числовой константой D4H. В зависимости от состояния флага переноса, программа либо продолжает свою работу, либо же возвращается на начало процедуры проверки появления синхроимпульса по метке LD-SYNC. Значит, сигнал пришел слишком рано и это не тот сигнал, который нужен.

```
40                CALL LD-EDGE-1     ;17
41                RET NC             ;11,5
```

После появления первого фронта синхроимпульса мы ожидаем появление второго. Это достигается за счет вызова процедуры LD-EDGE-1. Если после вызова данной процедуры, она возвращает значение флага переноса выключенным, то мы осуществляем возврат в вызвавшую программу по команде RET NC.

```
42                LD A,C             ;4
43                XOR #03            ;7
44                LD C,A             ;4
```

После появления обоих фронтов синхроимпульса необходимо осуществить соответствующую индикацию на экране. Это делается подготовкой в регистре C сигнализации о появлении следующего фронта синим и желтым цветами. Для этого мы загружаем в аккумулятор содержимое регистра C и осуществляем операцию исключающие "ИЛИ" над содержимым аккумулятора. После этого полученное значение вновь возвращается в регистр C.

```
45                LD H,#00           ;7
46                LD B,#80           ;7
47                JR LD-MARKER       ;12
```

В регистре H создается байт четности, а в регистр B помещается временная константа для считывания первого бита первого байта, описывающего тип читаемого

файла.

Как Вы помните, значение первого байта каждого файла характеризует тип файла и должно равняться нулю для заголовка и 255 для блока данных.

```
48    LD-LOOP     EX AF,A'F'          ;4
49                JR NZ,LD-FLAG       ;12,7
```

К этому фрагменту программа возвращается после чтения целого байта, поскольку предыдущая часть программы закончилась безусловным переходом на метку LD-MARKER. В этом участке программы флаг Z употреблен для различения байта, определяющего тип файла. Здесь также учитывается флаг C, определяющий, происходит ли процесс чтения или верификации.

```
50                JR NC,LD-VERIFY     ;12,7
51                LD (IX+0),L         ;19
52                JR LD-NEXT          ;12
```

В ходе выполнения данных операций считанный байт помещается под адресом, находящихся в IX. Для этого мы загружаем в ячейку памяти, адрес которой находится в (IX+0), содержимое регистра L. После этого осуществляется безусловный переход на метку LD-NEXT.

```
53    LD-FLAG     RL C                ;4
54                XOR L               ;4
55                RET NZ              ;11,5
```

При входе в данную процедуру в регистре A находится тип файла, который мы хотим считать, а в регистре L - первый считанный байт. Первая команда служит для временного хранения значений флага C в регистре C. Если считанный байт не совпадает с содержимым регистра A, то происходит возврат из процедуры со сброшенными флагами C и Z, что сигнализирует об ошибке.

```
56                LD A,C              ;4
57                RRA                 ;4
58                LD C,A              ;4
59                INC DE              ;4
60                JR LD-DEC           ;12
```

В случае совпадения типов восстанавливается состояние флага C и регистра C. Для этого мы загружаем содержимое регистра C в аккумулятор, осуществляем ротацию C через флаг переноса и возвращаем полученное значение из аккумулятора в регистр C. Увеличение регистра DE на единицу служит для компенсации его состояния при переходе к инструкции DEC DE при выполнении безусловного перехода на метку LD-DEC.

```
64    LD-NEXT     INC IX              ;10
65    LD-DEC      DEC DE              ;6
66                EX AF,A'F'          ;4
```

Данные команды устанавливают регистры IX и DE в состояние, необходимое для правильной работы программ. После этого осуществляется сохранение флага C флагового регистра F в безопасном месте с помощью использования команды перехода к набору альтернативных регистров микропроцессора Z-80.

```
67                LD B,#B2            ;7
68    LD-MARKER   LD L,01             ;7
```

После установки временной константы B2H в регистре B, обнуляется регистр L и его

младший бит устанавливается в единицу. После следующих сдвигов влево он будет постепенно перемещаться по направлению к флагу C и в конце концов окажется во флаге переноса, сигнализируя считывание всего байта.

```
69    LD-8-BITS   CALL LD-EDGE-2     ;17
70                RET NC             ;11,5
```

Данная процедура определяет время между двумя очередными фронтами сигналов. Для этого вызывается процедура LD-EDGE-2, которая работает в соответствии со значением временной константы занесенной в регистр B.

```
71                LD A,#CB           ;7
72                CP B               ;4
73                RL L               ;4
```

Если пара фронтов найдена за время, меньшее чем около 2400 тактов относительно вызова LD-EDGE-2, то они представляют собой ноль, а если за большее время, то единицу. Значение, получающееся после анализа информации, несет в себе флаг C.

Для правильного выполнения заданной процедуры мы загружаем в аккумулятор константу CBH, относительно которой ведется проверка содержимого регистра B.

```
74                LD B,#B0           ;7
75                JP NC,LD-8-BITS    ;10
```

После загрузки временной константы для следующего бита исследуется регистр B, который устанавливается только после считывания и размещения в регистре L всех восьми битов данного байта.

```
76                LD A,H             ;4
77                XOR L              ;4
78                LD H,A             ;4
```

Этот участок программы осуществляет модификацию байта четности, о котором мы уже писали ранее.

```
79                LD A,D             ;4
80                OR E               ;4
81                JR NZ,LD-LOOP      ;12,7
```

Здесь мы проверяем все ли байты были считаны и, если должны быть считаны еще какие-либо байты, то мы возвращаемся к метке LD-LOOP. Эта проверка осуществляется с помощью сравнения содержимого регистра DE с нулем и осуществлением операции условного перехода в зависимости от состояния флага Z флагового регистра.

```
82                LD A,H             ;4
83                CP #01             ;7
84                RET                ;10
```

После считывания последнего байта (четности), в регистре H должен быть ноль. Если это так, то флаг C включается, сигнализируя этим правильность окончания загрузки файла.

Так работают процедуры загрузки файлов с ленты в память компьютера. Надеемся, что этот материал окажется полезным для Вас. Тем не менее, его невозможно рассматривать в отрыве от информации по вопросам записи файлов на магнитную ленту в формате "Спектрума". Поэтому в следующей главе мы рассмотрим более подробно этот вопрос.

Глава 2. Стандартные процедуры записи информации на магнитную ленту.

Теперь мы переходим к рассмотрению процедур, реализующих запись информации на магнитную ленту и помещенных в ПЗУ компьютера.

Начнем с процедуры SA-BYTES, служащей для записи последовательности битов на кассету. Предполагается, что в момент ее запуска в регистре DE находится длина записываемого блока, в регистре IX - адрес первого байта, а в аккумуляторе - число, определяющее тип загружаемого блока. Напомним, что для заголовка этот байт равен нулю, в то время как для блока данных он равен 255. Первая процедура записи информации на магнитную ленту начинается с адреса 04C2H. Это является ее отправной точкой.

```
1       SA-BYTES    LD HL,#C53F         ;10
2                   PUSH HL             ;11
3                   LD HL,#1F80         ;10
4                   BIT 7,A             ;8
5                   JR Z,SA-FLAG        ;12,7
6                   LD HL,#0C98         ;10
```

Первый делом загружаем в регистр HL адрес процедуры возврата к БЕЙСИКу - SA-LD-RET, после чего данное значение сохраняется на стеке. Далее мы загружаем в регистр HL временные константы 1F80H и 0C98H, которые определяют длительность пилотирующего сигнала. Эта длительность составляет пять секунд для заголовка, и две секунды для блока данных. Определить, какой тип файла мы сейчас записываем на ленту, компьютеру помогает седьмой бит аккумулятора. Напомним, что в аккумуляторе содержится число, определяющее тип блока.

```
7       SA-FLAG     EX AF,A'F'          ;4
8                   INC DE              ;6
9                   DEC IX              ;10
10                  DI                  ;4
11                  LD A,#02            ;7
12                  LD B,A              ;4
```

После обработки блока содержимое регистра DE увеличивается на единицу. На столько же уменьшается и содержимое регистра IX. Все это делается после вызова альтернативного набора регистров микропроцессора и дает возможность определить байт как первый в записываемом блоке.

Выключение маскируемого прерывания необходимо, чтобы программа не была прервана для опроса клавиатуры (в обычном режиме опрос клавиатуры происходит пятьдесят раз в секунду, так как именно с такой регулярностью поступают импульсы на вход INT микропроцессора). Значение 2 в аккумуляторе установит предварительное напряжение в гнезде MIC на низкий уровень и цвет рамки на красный.

```
13      SA-LEADER   DJNZ SA-LEADER     ;13,8
14                  OUT (#FE),A         ;11
15                  XOR #0F             ;7
16                  LD B,#A4            ;7
17                  DEC L               ;4
18                  JR NZ,SA-LEADER     ;12,7
```

Следующий блок программы генерирует пилотирующий сигнал. Цикл SA-LEADER гарантирует задержку между последующими сменами напряжения в гнезде MIC. Затем, после установки напряжения в этом гнезде, четыре младших бита аккумулятора меняются на противоположные путем выполнения операции исключавшее "ИЛИ" над содержимым аккумулятора. Это подготавливает изменение напряжения на разъеме MIC на противоположное и смену цвета бордюра с красного на голубой или наоборот. Затем

уменьшается младший байт счетчика импульсов L.

Многие пользователи "Спектрума" наверняка встречали компьютерные программы без характерных полос на бордюре, присущих стандартным процедурам загрузки. Это объясняется тем, что цвет бордюра в ходе загрузки файла не изменяется. Как Вы помните, за цвет бордюра ответственны три младшие бита компьютерного порта 254. Поэтому, если Вы будете проводить операцию исключающее "ИЛИ" только над битом, ответственным за состояние сигнала на разъеме MIC и не будете затрагивать биты цвета бордюра. Вам также удастся добиться аналогичного эффекта.

```
19              DEC B             ;4
20              DEC H             ;4
21              JP P,SA-LEADER    ;10
```

После обнуления младшего байта счетчика импульсов L уменьшается и старший байт H и модифицируется в соответствующей ситуации временная постоянная в регистре B с целью учета дополнительно выполненных инструкций (в данном случае это тринадцать тактов микропроцессора). Как Вы уже могли заметить, после некоторых команд стоит несколько значений, характеризующих длительность выполнения команды в циклах микропроцессора Z-80. Это объясняется тем, что, команда может быть выполнена по-разному в зависимости от того, выполняется условие или нет. И, следовательно, длительность ее выполнения для каждого из этих случаев различна. Это объясняет тот факт, что в ходе работы такого рода процедур необходимо учитывать время, затрачиваемое на дополнительно выполняемые инструкции.

```
22              LD B,#2F          ;7
23    SA-SYNC1  DJNZ SA-SYNC1     ;10,8
24              OUT (#FE),A       ;11
```

После того, как был выслан пилотирующий сигнал, на порт MIC идет посылка переднего фронта синхроимпульса. Задержка в цикле SA-SYNC1 составляет 667 тактов, после чего содержимое аккумулятора передается в порт 254.

```
25              LD A,#0D          ;7
26              LD B,#37          ;7
27    SA-SYNC2  DJNZ SA-SYNC2     ;10,8
28              OUT (#FE),A       ;11
```

После 735 тактов идет высылка заднего фронта синхроимпульса, устанавливающего цвет бордюра голубым, напряжение на разъеме MIC - высоким. Изменение цвета и управляющего напряжения осуществляется занесением в аккумулятор числовой константы 0DH. Задержка осуществляется загрузкой в регистр B временной константы 37H. После выполнения задержки в цикле SA-SYNC2, значение аккумулятора пересылается в компьютерный порт 254.

```
29              LD BC,#3B0E       ;10
30              EX AF,A'F'        ;4
31              LD L,A            ;4
32              JP SA-START       ;10
```

Загрузка константы 3B0EH в регистр BC представляет собой загрузку двух различных чисел в отдельные регистры данной регистровой пары. 3BH - это временная постоянная, загружаемая в регистр B, а 0EH - придает начальное значение регистру C, в котором будет сохраняться состояние последнего напряжения в гнезде MIC и цвет бордюра (начальное состояние низкое напряжение и желтый цвет бордюра). Воспроизведенный тип блока помещается в регистр L и посылается на ленту как первый байт данного блока. После этого осуществляется безусловный переход на метку SA-START.

```
33    SA-LOOP     LD A,D              ;4
34                OR E                ;4
35                JR Z,SA-PARITY      ;10,7
```

Процедура SA-LOOP - это начало главного цикла, записывающего следующий байт. Если счетчик байтов, который организуется в регистре DE, достиг нуля, то остается записать только байт четности. Проверка на ноль осуществляется типичным способом, характерным для микропроцессора Z-80. В аккумулятор загружаем содержимое регистра D и, используя содержимое регистра E, выполняем операцию логическое "ИЛИ". Результат операции может быть равен нулю только в том случае, если оба байта равны нулю, поэтому нам только остается проверить содержимое флага Z флагового регистра процессора для точного установления результата выполнения данной операции.

```
36                LD L,(IX+0)         ;19
37    SA-LOOP-P   LD A,H              ;4
33                XOR L               ;4
39    SA-START    LD H,A              ;4
```

Данный отрезок программы служит для получения следующего байта для высылки его на магнитофон. Он также используется для запуска созданного в регистре H байта четности. Регистр H инициируется байтом, определяющим тип записываемого блока. (Методика построения байта четности в ходе операций записи и загрузки была нами рассмотрена в главе 1 данного тома).

```
40                LD A,1              ;7
41                SCF                 ;4
42                JP SA-8-BITS        ;10
```

Загружаемое в аккумулятор значение единицы установит напряжение в гнезде MIC на высокий уровень и цвет бордюра станет синим. Принудительное включение флага C флагового регистра F будет выполнять роль указателя, позволяющего утверждать, что выслано уже восемь битов данного байта.

В завершение процедуры мы осуществляем безусловный переход на метку SA-8-BITS.

```
43    SA-PARITY   LD L,H              ;4
44                JR SA-LOOP-P        ;12
```

Процедура SA-PARITY готовит высылку байта четности, как последнего байта, записываемого на магнитофонную ленту. Для этого в регистр L, который обычно используется для хранения высылаемого байта, записываем байт четности, который находится в регистре H. После этого осуществляется переход к процедуре SA-LOOP-P, которая, собственно, и осуществляет запись.

```
45    SA-BIT2     LD A,C              ;4
46                BIT 7,B             ;8
```

К процедуре SA-BIT2 мы обращаемся при втором проходе цикла, записывающего одиночный бит (это осуществляется перед высылкой заднего фронта сигнала). Проверка седьмого бита в регистре B имеет цель только обнуление флага C, сигнализируя таким образом второй проход.

```
47    SA-BIT1     DJNZ SA-BIT1        ;10,8
48                JR NC,SA-OUT        ;12,7
```

Следующая процедура формирует главный цикл задержки между последовательными фронтами сигналов. Флаг C несет в себе действительное значение высылаемого бита.

```
49              LD B,#42          ;7
50    SA-SET    DJNZ SA-SET       ;10,8
```

Перед высылкой логической единицы нам необходимо задержать работу на B55 дополнительных тактов. Для этого в регистр B мы заносим временную константу 42H, которая организует нам необходимый цикл задержки.

```
51    SA-OUT    OUT (#FE),A       ;11
52              LD B,#3E          ;7
53              JR NZ,SA-BIT2     ;12,7
```

Данная процедура организует высылку заднего фронта сигнала. Это происходит после высылки переднего фронта путем занесения в регистр B константы 3EH и вызова подпрограммы SA-BIT2.

```
54              DEC B             ;4
55              XOR A             ;4
56              INC A             ;4
57    SA-8-BITS RL L              ;8
58              JP NZ,SA-BIT1     ;10
```

После модификации временной константы в регистре B путем уменьшения ее на единицу, обнуляется флаг Z и в аккумулятор вносится значение единицы (бордюр становится синим, а напряжение в MIC - низкого уровня). Затем, после увеличения аккумулятора на единицу, во флаг C перемещается следующий бит из регистра L. Если регистр L отличен от нуля, то он знаменует собой начало цикла SA-BIT1, причем первый вход в этот цикл выполняется с установленным флагом C, в то время как все остальные - с обнуленным. Благодаря этому только после восьми проходов регистр L достигает значения нуля.

```
59              DEC DE            ;6
60              INC IX            ;10
61              LD B,#31          ;7
```

Данный отрезок программы осуществляет модификацию счетчика путем уменьшения на единицу содержимого регистра DE и увеличения на единицу содержимого регистра IX. Следующим этапом идет подготовка временной константы путем засылки в регистр B значения 31H.

```
62              LD A,#7E          ;7
63              IN A,(#FE)        ;11
64              RRA               ;4
65              RET NC            ;11,5
```

Здесь выполняется контроль нажатия клавиши BREAK и, если она нажата, то программа передает управление на адрес 053FH процедуре обработки возврата в БЕЙСИК SA/LD-RET.

Проверка нажатия клавиши начинается с чтения в аккумулятор значения из порта FEH. После получения значения осуществляется сдвиг аккумулятора вправо с использованием флага переноса, в который попадает нулевой бит аккумулятора. Этот бит и характеризует тот факт, была ли нажата клавиша BREAK. Завершает этот участок программы процедура условного возврата, которая, анализируя состояние флага переноса либо продолжает выполнение программы по нижеследующему маршруту, либо осуществляет возврат к процедуре выхода в БЕЙСИК.

```
66              LD A,D            ;4
67              INC A             ;4
68              JP NZ,SA-LOOP     ;10
```

Контроль счетчика байтов осуществляет предварительную проверку на достижение нуля в регистре D. Для этого мы передаем в аккумулятор значение регистра D, увеличиваем аккумулятор на единицу и проверяем содержимое регистра A на наличие нуля. Переход к метке SA-LOOP выполняется также в случае достижения регистром DE значения нуля, так как необходимо еще записать байт четности.

```
69              LD B,#3B          ;7
70    SA-DELAY  DJNZ SA-DELAY     ;10,8
71              RET               ;10
```

После достижения в регистре DE значения #FFFF и короткой паузы управление передается к процедуре возврата к БЕЙСИКУ SA/LD-RET. Как Вы знаете, в компьютере при вычитании значений из регистра, содержащего ноль, у нас получается значение FFFFH, FFFEH, FFFDH и так далее. Пауза организуется с помощью занесения в регистр временной константы 3BH и создания цикла на основе этого значения. Последний байт процедуры SA-BITS помещается в ячейке с адресом 053EH.

Мы с Вами рассмотрели стандартные процедуры считывания и записи данных на магнитную ленту. Осталось рассмотреть процедуру обработки возврата в БЕЙСИК.

Эта процедура имеет название SA/LD-RET и используется для выхода в БЕЙСИК из подпрограмм загрузки или записи в случае успешного или неуспешного их завершения, или же нажатие клавиши BREAK. Она необходима для того, чтобы гарантировать возврат к БЕЙСИКУ из процедуры записи/считывания, как в случае их правильного окончаний, так и в случае появления ошибок или прерывания оператора, то есть клавишей BREAK. Время выполнения инструкции для этой процедуры не существенно.

```
1     SA/LD-RET   PUSH AF
2                 LD A,(#5C48)
3                 AND #38
4                 RRCA
5                 RRCA
6                 RRCA
7                 OUT (#FE),A
```

Помещение флагового регистра на стек имеет целью сохранение флага C: обнуление его означает прерывание клавишей BREAK или ошибку чтения. Затем в аккумулятор загружается значение системной переменной BORDER и на основе ее третьего, четвертого и пятого битое воспроизводится цвет бордюра экрана, который был перед запуском соответствующей процедуры, обслуживающей магнитофон. Это осуществляется путем высылки содержимого аккумулятора в компьютерный порт 254.

```
8                 LD A,#7E
9                 IN A,(#FE)
10                RRA
11                EI
12                JR C,SA/LD-END
```

В этом отрезке программы еще раз проверяется, была ли нажата клавиша BREAK для того, чтобы можно было выдать соответствующее информационное сообщение. После этого мы включаем маскируемое прерывание и осуществляем условный переход в зависимости от состояния флага переноса C, который изменяется при нашей проверке нажатия клавиши BREAK. В случае, если клавиша BREAK была действительно нажата, мы осуществляем переход к процедуре SA/LD-END.

```
13    REPORT-D    RST #08
14                DEFB #0D
```

Эти команды вызывают возврат к БЕЙСИКУ с выдачей сообщения:

```
D-BREAK-CONT REPEATS.
```

Данное сообщение, как Вы помните, выдается, если программа прерывается оператором с помощью клавиши BREAK. Это только один из типов сообщений, которые могут выдаваться в подобном случае. Оно говорит, что клавиша BREAK была нажата во время действия периферийной операции. Действие CONTINUE после этого оператора обычное.

```
15     SA/LD-END   POP AF
16                 RET
```

Первым делом мы извлекаем из стека содержимое флагового регистра. Это необходимо для воспроизведения флага переноса и правильного возврата к процедуре запуска. Следует отметить, что последний байт процедуры SA/LD-RET занимает ячейку за адресом 0555H.

Мы с Вами закончили рассмотрение процедур чтения и записи на магнитную ленту в формате "Спектрума". Данный материал мы старались построить так, чтобы он был легко доступен для начинающих и чтобы в то же время и профессионал мог почерпнуть в нем что-то новое.

В то же время, это был вынужденный шаг, поскольку цель, которую мы ставили перед собой - освоить защиту программ от копирования была бы не достигнута, если бы Вы не смогли разобраться в принципах действия стандартных загрузочных процедур, зашитых в ПЗУ "Спектрума". И, тем не менее, знание одной лишь работы стандартных подпрограмм Вам будет явно недостаточно для написания мощных защит от копирования. В следующей главе мы рассмотрим конкретную программу защиты от копирования информации на магнитной ленте. Это лишь один из многих способов защиты от копирования и мы надеемся, что читатель отнесется к изучению нашего материала творчески, то есть постарается не только вникнуть в суть публикации, но и разработает свои собственные более мощные программы.

Глава 3. Методы защиты от копирования для ПЭВМ "Спектрум".

1. Методика создания оригинальной процедуры записи информации на магнитную ленту.

Информация, приведенная в предыдущих главах, поможет Вам поближе познакомиться со структурой хранения информации на магнитной ленте в формате "Спектрума". Однако, этого еще не достаточно для написания мощных программ защиты от копирования и тем более для создания оригинальных защищающих процедур, поскольку необходимо предварительно ознакомиться с основными существующими приемами такой защиты.

Как Вы помните, время, которое затрачивает процессор на обработку одной машинной операции несоизмеримо мало по сравнению с минимально допустимым временем, за которое можно закодировать один бит информации на магнитной ленте. Поэтому для того, чтобы правильно записать информацию на магнитный носитель, процессор вынужден делать временные задержки, оформляемые в программе в виде циклов с предварительно заданными временными константами, хранящимися в регистре B.

Таким образом, одно из направлений в создании собственной защиты - это изменение временных констант в сторону незначительного уменьшения или увеличения их значений, что позволяет сделать непригодными стандартные процедуры чтения данных, записанных в новом формате.

Для того, чтобы обеспечить надежную защиту, нет необходимости в полном изменении всех временных констант - на практике достаточно изменения одной из них. Мы можем изменить временную константу задержки, характерную для стандартного пилотирующего сигнала, стандартного синхроимпульса, а также характерную для

стандартной записи логического нуля или единицы. Все это приведет к тому, что созданная и выгруженная нами программа не сможет правильно считываться стандартными процедурами, встроенными в ПЗУ компьютера.

Для того, чтобы облегчить Вашу работу по созданию собственных защит, мы хотим предложить Вам достаточно удобный способ создания программ подобного рода. Если Вы внимательно читали предыдущие главы, то наверняка обратили внимание на то, что процедуры записи и чтения состоят из большого числа подпрограмм, которые периодически вызываются в ходе работы основной процедуры. Для создания своей процедуры записи и чтения Вам фактически необходимо лишь создать головную программу, которая внесет необходимые изменения во временные константы. Остальные подпрограммы можно использовать из того стандартного набора, который уже "зашит" в ПЗУ.

Возможно, что этот путь устроит не всех и кто-то захочет написать собственные процедуры записи и чтения программ в полном объеме самостоятельно. Что ж, мы не ставим своей целью навязывать Вам свои приемы программирования, а лишь хотим ознакомить Вас с основными методами защиты.

Итак, мы предлагаем Вам для рассмотрения достаточно простой способ защиты, а именно: изменение временной константы задержки для пилотирующего сигнала. Создание собственного, отличного от стандартного, пилотирующего сигнала, позволило бы Вам защитить свою программу от несанкционированного копирования уже в начальный момент загрузки. Это объясняется тем, что все программы, записанные на магнитную ленту в формате Спектрума начинаются именно с пилотирующего сигнала.

Для создания защищенной от копирования программы, Вам необходимо исключить возможность загрузки данной программы в копировщик. Поскольку копировщик использует стандартные процедуры записи и чтения информации с магнитной ленты, то Вам необходимо блокировать его работу путем создания файла, записанного в одном из созданных Вами форматов, отличном от стандартного формата "Спектрума". Чтобы защитить программу от копирования, на практике нет необходимости в полном изменении формата записи всех блоков данных, хранимых на магнитной ленте. Для этого достаточно записать в измененном формате всего лишь один небольшой блок данных, который хранит в себе информацию, без которой не может начаться выполнение программ.

Поскольку данный файл записан в измененном формате, то его невозможно будет скопировать с помощью копировщика, а так как он несет в себе жизненно важные сведения, необходимые для правильной работы программы, то без него программа не сможет нормально функционировать, даже если попытаться отбросить его в процессе копирования.

Рассмотрим принцип формирования пилоттона в формате "Спектрума".

```
SA-BYTES        LD HL,#053F
                PUSH HL
                LD HL,#1F80
                BIT 7,A
                JR Z,SA-FLAG
                LD HL,#0C98
SA-FLAG         EX AF,A'F'
                INC DE
                DEC IX
                DI
                LD A,#02
                LD B,A
SA-LEADER       DJNZ SA-LEADER
                OUT (#FE),A
                XOR #0F
                LD B,#A4
                DEC L
                JR NZ,SA-LEADER
                DEC B
                DEC H
                JP P,SA-LEADER
```

Перед входом в данную процедуру предполагается, что регистры хранят информацию о записываемом файле. В регистре DE находится длина записываемого блока, в регистре IX - адрес первого байта, а в аккумуляторе - число, которое определяет тип записываемого блока, причем ноль выставляется для заголовка, а 255(0FFH) - непосредственно для блока данных.

После начала работы процедуры SA-BITS идет подготовка к формированию пилотирующего сигнала. В регистр HL могут быть занесены два числа, определяющие длительность пилотирующего сигнала. 1F80H заносится в регистр HL в случае записи заголовка - и это соответствует приблизительно пяти секундам звучания пилотирующего сигнала, а значение 0C98H заносится в данный регистр в случае записи непосредственно блока данных, что соответствует приблизительно двум секундам звучания пилотирующего сигнала. Какой тип значения заносить в регистр HL определяется путем проверки содержимого аккумулятора, точнее, его седьмого бита.

Формирование пилотирующего сигнала осуществляется процедурой SA-LEADER. Временная задержка осуществляется с помощью занесения в регистр B значения A4H. Таким образом, путем изменения содержимого аккумулятора, мы достигаем периодического перепада напряжения на выходном разъеме, связывавшем Спектрум с магнитофоном и служащим для записи информации. Если в этой процедуре значение, заносимое в регистр B, то Вам удастся сформировать свой собственный пилотирующий сигнал, отличный от стандартного, который позволит защитить Вашу программу от копирования.

После формирования нестандартного пилотирующего сигнала остается продолжить запись Вашей программы на магнитную ленту. Одним из способов осуществления этого является переход к стандартной подпрограмме SA-SYNC1, которая, если Вы правильно оформили свою процедуру записи, то стандартная подпрограмма сама закончит работу по сохранению информации на магнитном носителе и возвратится в операционную среду "Спектрума".

Теперь Вы видите, что нет ничего сложного в том, чтобы составить свою собственную программу по записи информации на магнитофон в формате "Спектрума". Для этого достаточно лишь немного модернизировать одну из стандартных процедур записи, в нашем случае SA-LEADER. Остальные процедуры записи можно использовать стандартные и, лишь когда Вы более подробно познакомитесь со структурой встроенных процедур, Вы сможете начать писать собственные программы дальнейшей обработки информации и возврата в операционную среду. Ми предлагаем Вам начинать с простейшего и постепенно совершенствовать свои приемы программирования.

2. Методика создания оригинальной процедуры чтения информации с магнитной ленты.

Мы с Вами рассмотрели методику создания оригинальных процедур записи информации на магнитофон, используемую при защите информации от копирования. При ее рассмотрении Вам был предложен метод, позволяющий ускорить подобного рода разработки за счет использования встроенных в "Спектрум" процедур. Однако, такой метод не всегда приемлем, и поэтому, приступая к рассмотрению процедуры чтения информации с магнитной ленты, мы остановим свой выбор на создании полноценной процедуры выполняющей все функции чтения информации с магнитофона.

Процедуры чтения информации с магнитной ленты, являются более сложными в сравнении с аналогичными процедурами записи. Это объясняется тем, что основа действия процедур заключается в многократной проверке шестого бита порта 254 и в определение интервала между двумя следующими друг за другом фронтами. Таким образом, при чтении блока с ленты необходимым является точное определение интервалов между фронтами сигналов. Работу по определению этих интервалов выполняет вспомогательная процедура LD-EDGE-2, размещенная по адресу 05E3H в ПЗУ "Спектрума". Эта точка входа в ПЗУ используется, когда необходимо установить, сколько времени прошло от момента вызова

процедуры до обнаружения двух, следующих одна за другим смен уровней сигналов в гнезде EAR и на сколько обе эти смены уровней сигнала соответствуют по времени допустимым пределам требуемых временных интервалов.

Второй важной точкой входа в ПЗУ является стандартная процедура LD-EDGE-1, расположенная по адресу 05E7H. Она позволяет определить, сколько времени прошло с момента вызова процедуры до извлечения первого фронта сигнала. Вызывается она после помещения в регистр B временной константы, ограничивающей количество проверок порта FEH. Превышение допустимого времени ожидания смены сигнала или определение нажатия клавиши BREAK сопровождается обнулением флага переноса. Для распознавания каждой из этих ситуаций служит флаг нуля. Если флаг Z обнулен, то в ходе работы программы была нажата клавиша BREAK, а если он включен, то произошло превышение временного интервала.

В рассматриваемой нами методике изменения стандартного пилотирующего сигнала Вам необходимо как можно более точно рассчитать необходимое время задержки для четкого поиска фронтов модернизированного пилоттона. Как мы и обещали, ниже приводится специализированная программа для загрузки с ленты файла, начинающегося с модернизированного пилоттона.

```
                ORG 50000
                LD DE,256
                LD IX,23296
                LD A,255
1     LD-BYTES  INC D             ;4
2               EX AF,A'F'        ;4
3               DEC D             ;4
4               DI                ;4
5               LD A,#0F          ;7
6               OUT (#FE),A       ;11
7               LD HL,#053F       ;10
6               PUSH HL
9               IN A,(#FE)        ;11
10              RRA               ;4
11              AND #20           ;7
12              OR #02            ;7
13              LD C,A            ;4
14              CP A              ;4
15    LD-BREAK  RET NZ            ;11,7
16    LD-START  CALL LD-EDGE-1    ;17
17              JR NC,LD-BREAK    ;12,7
18              LD HL,#0415       ;10
19    LD-WAIT   DJNZ LD-WAIT      ;10,6
20              DEC HL            ;6
21              LD A,H            ;4
22              OR L              ;4
23              JR NZ,LD-WAIT     ;12,7
24              CALL LD-EDGE-2    ;17
25              JR NC,LD-BREAK    ;12,7
26    LD-LEADER LD B,#9C          ;7
27              CALL LD-EDGE-2    ;17
28              JR NC,LD-BREAK    ;12,7
29              LD A,#C6          ;7
30              CP B              ;4
31              JR NC,LD-START    ;12,7
32              INC H             ;4
33              JR NZ,LD-LEADER   ;12,7
34    LD-SYNC   LD B,#C9          ;7
35              CALL LD-EDGE-1    ;17
36              JR NC,LD-BREAK    ;12,7
37              LD A,B            ;4
38              CP #D4            ;7
39              JR NC,LD-SYNC     ;12,7
40              CALL LD-EDGE-1    ;17
41              RET NC            ;11,5
```

```
42              LD A,C              ;4
43              XOR #03             ;7
44              LD C,A              ;4
45              LD H,#00            ;7
46              LD B,#B0            ;7
47              JR LD-MARKER        ;12
48  LD-LOOP     EX AF,A'F'          ;4
49              JR NZ,LD-FLAG       ;12,7
50              JR NC,LD-VERIFY     ;12,7
51              LD (IX+C),L         ;19
52              JR LD-NEXT          ;12
53  LD-FLAG     RL,C                ;4
54              XOR L               ;4
55              RET NZ              ;11,5
56              LD A,C              ;4
57              RRA                 ;4
58              LD C,A              ;4
59              INC DE              ;4
60              JR LD-DEC           ;12
64  LD-NEXT     INC IX              ;10
65  LD-DEC      DEC DE              ;6
66              EX AF,A'F'          ;4
57              LD B,#B2            ;7
68  LD-MARKER   LD L,01             ;7
69  LD-8-BITS   CALL LD-EDGE-2      ;17
70              RET NC              ;11,5
71              LD A,#CB            ;7
72              CP B                ;4
73              RL L                ;4
74              LD B,#B0            ;7
75              JP NC,LD-8-BITS     ;10
76              LD A,H              ;4
77              XOR L               ;4
78              LD H,A              ;4
79              LD A,D              ;4
80              OR E                ;4
81              JR NZ,LD-LOOP       ;12,7
82              LD A,H              ;4
63              CP #01              ;7
84              RET                 ;10

1   LD-EDGE-2   CALL LD-EDGE 1      ;17
2               RET NC              ;11,5
3   LD-EDGE-1   LD A,#16            ;7
4   LD-DELAY    DEC A               ;4
5               JR NZ,LD-DELAY      ;12,7
6               AND A               ;4
7   LD-SAMPLE   INC B               ;4
8               RET Z               ;11,7
9               LD A,#7F            ;7
10              IN A,(#FE)          ;11
11              RRA                 ;4
12              RET NC              ;11,7
13              XOR C               ;4
14              AND #20             ;7
15              JR Z,LD-SAMPLE      ;12,7
16              LD A,C              ;4
17              CPL                 ;4
18              LD C,A              ;4
19              AND #07             ;7
20              OR #08              ;7
21              OUT (#FE),A         ;11
22              SCF                 ;4
23              RET                 ;10
```

Приведенная выше программа полностью соответствует стандартной процедуре загрузки информации и может осуществлять чтение файлов в формате "Спектрума". Если Вы внимательно присмотритесь к данной программе в машинных кодах, то легко обнаружите, что за загрузку пилоттона ответственна лишь одна процедура LD-LEADER (в нашей программе ей соответствуют строки 26-31). Здесь временная константа 9CH позволяет процедуре LD-EDGE-2 искать два фронта сигнала за время, не превышающее 7000 тактов работы процессора. С другой стороны, после их нахождения проверяется, прошло ли с момента вызова процедуры LD-EDGE-2 до появления заднего фронта не менее, чем 3366 тактов работы процессора. Этим способом отличаются фронты пилотирующего сигнала от значительно более близких пар фронтов информационных сигналов.

Загружаемая в регистр B временная константа 9CH перед вызовом процедуры LD-EDGE-2 формирует необходимую задержку для правильной работы этой стандартной процедуры. После того, как мы возвращаемся из процедуры LD-EDGE-2, мы первым делом проверяем, была ли нажата клавиша BREAK и лишь в следующий момент сравниваем содержимое регистра B с предварительно загруженной в аккумулятор константой C6H. Команда сравнения, применяемая в этом случае, позволяет нам сравнить содержимое аккумулятора с числом, содержащимся в регистре B. В результате операции числовые значения не изменяются. Результат сказывается только на флагах регистра F. Если содержимое аккумулятора больше сравниваемого числа или равно ему, то флаг переноса будет выключен. Если же оно меньше, то флаг переноса будет равен единице.

Как Вы помните, процедура LD-EDGE-2 позволяет установить, сколько времени прошло от момента ее вызова до обнаружения двух, следующих друг за другом смен уровней сигналов в магнитофонном гнезде. Для этого в регистр B заносится константа, определяющая количество проверок магнитофонного гнезда EAR. Если изменение фронта не обнаружено, то мы выходим из процедуры LD-EDGE-2 по обнулению регистра B. Если же мы обнаружили смену фронтов в тот момент, когда регистр B еще не достиг своего нулевого значения, мы выходим из вышеуказанной процедуры и сравниваем оставшееся в регистре B значение с константой, загружаемой в аккумулятор. Это необходимо для того, чтобы проверить, прошло ли с момента вызова LD-EDGE-2 до появления заднего фронта не менее, чем 3366 тактов работы процессора.

Для того, чтобы создать собственную процедуру чтения файлов с магнитофона, Вам необходимо рассчитать, какие значения заносить в регистр B перед вызовом LD-EDGE-2 из подпрограммы LD-LEADER и с каким значением аккумулятора необходимо сравнивать остаток регистра B после отыскания фронта сигнала.

Все вышесказанное касается лишь того случая, когда Вы будете использовать метод модернизации пилотирующего сигнала, чтобы добиться его отличия от стандарта. Для каждого варианта такого сигнала Вам необходимо создавать собственную оригинальную процедуру загрузки, то есть подсчитывать время задержки.

Мы с Вами рассмотрели методику защиты от копирования с использованием нестандартного пилотирующего сигнала. Однако самой важной целью этой работы было показать Вам, что Вы в состоянии сами написать как процедуру записи программы на магнитную ленту, так и процедуру чтения. Это еще ближе знакомит Вас с работой компьютера, помогает преодолеть естественную неуверенность перед машинным кодом и, как мы надеемся, поможет Вам реализовать по новому свои способности в программировании.

СЕКРЕТЫ TRDOS

Алексеев А.Г.

Итак, Вы обзавелись БЕТА-ДИСК интерфейсом, а также чистыми дискетами. Если есть у кого переписывать готовые программы на дискетах - Вы на некоторое время будете избавлены от прочих забот, наслаждаясь тем, как лихо проходит загрузка игр. Вы даже поначалу не будете обращать внимание на то, что где-то подпорчена заставка (нет двух нижних строк), а где-то на экран загружается какая-то цветная дрянь. Правда, после этого игра стартует нормально, но выглядит это как-то кустарно. Кроме того, наверняка есть у Вас игры на магнитной ленте, которые Вы относите к самим "игручим", а их дисковых версий как-то не оказалось у Ваших друзей. В общем, постоянные "житейские" заботы будут каждый день подталкивать Вас на более глубокое изучение TR-DOS. Поэтому я поделюсь тем опытом, который накопился у меня с момента подключения БЕТА-диска.

Вам предлагается обобщенный материал, собранный из разрозненных источников, а также уникальные данные, полученные в результате экспериментальной работы.

Начальные сведения по дисковой операционной системе TR-DOS достаточно подробно изложены в многочисленных описаниях, распространяемых в достаточном количестве на компьютерных рынках страны. Поэтому в этой статье я буду подразумевать, что владельцы БЕТА-ДИСК интерфейсов уже знакомы с этими материалами.

Первое, что усваивает каждый пользователь TR-DOS, это то, что вместо, например:

```
LOAD "" CODE
```

в Бейсик-строке теперь надо ставить:

```
RANDOMIZE USR 15619 : REM : LOAD "name" CODE
```

а вход в систему TR-DOS из Бейсика производится командой:

```
RANDOMIZE USR 15616
```

Прежде всего, давайте выясним что это за адреса? 15616 и 15619 это же ведь адреса ПЗУ, того самого, которое было до подключения TR-DOS. Не вступаем ли мы в конфликт с машинный кодом основного ПЗУ Спектрума? Оказывается нет. С адреса 15616 в ПЗУ Спектрума расположен символьный набор, начиная с пробела, то есть, в нормальном режиме работы "Спектрума" никогда не происходит выполнение машинного кода в этих адресах. В случае БЕТА-ДИСК интерфейса, при попадании на этот адрес срабатывает селектор адреса, который подключает теневое ПЗУ и дальнейшая работа происходит в нем. В теневом ПЗУ с системой TR-DOS по этим адресам располагаются команды, передающие управление в соответствующие точки ПЗУ TR-DOS.

Рассмотрим теперь более детально, как же устроена TR-DOS.

КАРТА ПАМЯТИ ОЗУ.

Те сведения, которые изложены в "фирменных" руководствах по TR-DOS, можно охарактеризовать как TR-DOS снаружи. Ну, например, в "фирменном" описании сказано, что для работы TR-DOS выделяется дополнительно 112 байт ОЗУ. И еще говорится о том, что в процессе работы TR-DOS динамически выделяется 256 байт, причем происходит это незаметно для пользователя. И все про это. Попробуем раскрыть поподробнее эти фразы, взглянув на TR-DOS изнутри.

Для начала посмотрим, где эти 112 байт ОЗУ выделяются. До инициализации БЕТА-ДИСК интерфейса карта памяти ОЗУ имела следующий вид (только то, что представляет интерес):

```
-----------------------
Экранная память   16384

-----------------------
Буфер принтера    23296

-----------------------
Системные         23552
переменные
-----------------------
Бейсик-программа  23755 <-- PROG

----------------------- <--RAMTOP
Свободно
                  65535 <-- PRAMT
-----------------------
```

При подаче команды RANDOMIZE USR 15616 происходит инициализация БЕТА-ДИСК интерфейса. При этом перестраивается карта памяти. Дополнительные 112 байт ОЗУ выделяется перед областью Бейсик-системы. Системная переменная PROG становится равной 23755+112=23867. Бейсик-программа со всеми Бейсик-переменными и маркером конца передвигается на 112 байт ближе к RAMTOP. А область, отведенная под Бейсик-систему становится, таким образом, меньше на 112 байт:

```
-----------------------
Экранная память   16384

-----------------------
Буфер принтера    23296

-----------------------
Системные         23552
переменные
- - - - - - - - - - - -
Системные
переменные TR-DOS
-----------------------
Бейсик-программа  23867 <-- PROG

----------------------- <--RAMTOP
Свободно
                  65535 <-- PRAMT
-----------------------
```

Кроме того, непосредственно в тот момент, когда происходит чтение или запись сектора данных, Бейсик-программа отодвигается еще дальше к RAMTOP. Эту переброску осуществляет TR-DOS и происходит она для пользователя незаметно (однако это может доставить массу проблем при адаптации магнитофонных программ под диск). Бейсик-программа со всеми своими переменными отодвигается еще на 257 байт. Это место освобождается непосредственно для операции чтения-записи сектора данных на диск: 256 байт информации плюс один байт - контрольная сумма сектора. Карта памяти в этот момент имеет следующий вид:

```
-----------------------
Экранная память    16384

-----------------------
Буфер принтера 23296

-----------------------
Системные          23552
переменные
- - - - - - - - - - - - -
Системные
переменные TR-DOS
- - - - - - - - - - - - -
257-байтное
пространство
-----------------------
Бейсик-программа  24124 <-- PROG

----------------------- <--RAMTOP
Свободно
                  65535 <-- PRAMT
-----------------------
```

По завершении операции чтения-записи Бейсик-программа возвращается на свое прежнее место - в адрес 23867.

Такое изменение адресов не то что происходит незаметно, но, как уже говорилось, может создать массу проблем для адаптации магнитофонных версий программ под диск. Большинство пользователей Спектрумов начинали с магнитофонных вариантов и БЕТА-диском оснащали свои уже готовые компьютеры. Поэтому, конечно, одной из первых потребностей, с которой Вы сталкиваетесь, (и с которой столкнулся я), является адаптация своих собственных программ под БЕТА-диск. Переделка фирменных игровых программ - это тема для отдельного разговора, однако на рынке существует огромное множество дисковых версий игр. Дешевле пойти и купить то, что надо. Но свои собственные программы Вам придется переделывать самим. Проблемы могут быть вызваны, например, нехваткой памяти в связи с выделением дополнительных 112 байт ОЗУ. Если значение RAMTOP небольшое, порядка 25000, то под Бейсик-программу остается совсем мало места. И программа, которая нормально работает в магнитофонном варианте, может отказаться работать в дисковом. Значение RAMTOP изменить бывает проблематично, поэтому приходится прибегать к другим приемам по экономии памяти. Многие из этих приемов достаточно подробно разбирались на страницах ZX-РЕВЮ. Это, в частности, замена цифровых значений на VAL "..." и другие приемы. Может оказаться так, что в принципе Бейсик-программа работает, но если в программе встречаются команды TR-DOS с префиксом:

```
RANDOMIZE USR 15619: REM :
```

то вместо выполнения их, на экране появляется сообщение об ошибке: "Out of memory". Это значит, что свободного места не хватает при попытке выделить 257-байтное пространство для операции чтения-записи сектора.

Если способы экономии памяти исчерпаны, то возможно придется серьезно подумать над изменением структуры программы в целях увеличения значения RAMTOP. Кроме этого, следует упомянуть еще об одной проблеме. Во многих программах применяется нулевая строка для размещения в ней блоков машинных кодов за оператором REM. Об этом подробно рассказывалось в ZX-РЕВЮ N 9-10 за 1992 год, см. стр.193. Там говорилось, что если строка с REM это начальная строка программы, то свободное место за REM начинается с адреса PROG+5. В случае TR-DOS, это будет 23867+5=23872. Однако, если Вы попробуете загрузить с диска блок кодов по этому адресу при помощи команды TR-DOS:

```
LOAD "name" CODE 23872
```

то у Вас ничего не получится. Дело вот в чем. В тот момент когда происходит загрузка (выполнение процедуры чтения сектора данных), Бейсик-програма и оператор REM вместе с ней уже имеют другой адрес - на 257 больший, чем тот, который был. Поэтому загрузку кодов с диска в нулевую строку надо выполнять следующим образом:

```
LOAD "name" CODE (23872+257)
```

Тогда блок кодов попадет точно в адрес 23872.

Вышесказанное относится к загрузке блока кодов и адреса между PROG и RAMTOP. То же относится и к записи блока кодов из этого интервала адресов на диск.

СИСТЕМНЫЕ ПЕРЕМЕННЫЕ TR-DOS.

Продолжая взгляд на TR-DOS изнутри, рассмотрим некоторые из системных переменных, составляющих дополнительные 112 байт ОЗУ. Еще раз упомяну: предполагается, что читатель знаком с описанием TR-DOS в начальном объеме. Сначала кратко.

23752 (#5CC8) - 1 байт - режим работы диска "A"
23753 (#5CC9) - 1 байт - режим работы диска "B"
23754 (#5CCA) - 1 байт - режим работы диска "C"
23755 (#5CCB) - 1 байт - режим работы диска "D"
23773 (#5CDD) - 8 байт - имя файла в кодах ASCII
23781 (#5CE5) - 1 байт - тип файла
23782 (#5CE6) - 2 байта - параметр START
23784 (#5CE8) - 2 байта - длина файла
23786 (#5CEA) - 1 байт - объем файла в секторах
23787 (#5CEB) - 1 байт - номер первого сектора файла
23788 (#5CEC) - 1 байт - номер первого трека файла
23798 (#5CF6) - 1 байт - дисковод для временной операции
23800 (#5CF8) - 1 байт - дисковод при операции с 2 файлами
23801 (#5CF9) - 1 байт - признак операции READ/VERIFY
23802 (#5CFA) - 1 байт - время перемещения головки дисковода "A"
23803 (#5CFB) - 1 байт - время перемещения головки дисковода "B"
23804 (#5CFC) - 1 байт - время перемещения головки дисковода "C"
23805 (#5CFD) - 1 байт - время перемещения головки дисковода "D"
23823 (#5D0F) - 1 байт - код ошибки TR-DOS
23830 (#5D16) - 1 байт - копия системного регистра
23833 (#5D19) - 1 байт - номер дисковода
23838 (#5D1E) - 1 байт - номер файла, если он найден.

Теперь подробнее

С адреса 23752 по 23755 расположены 4 числа, по одному на каждый из возможных четырех подключенных дисководов A, B, C и D. 7-й бит этого числа определяет количество дорожек дисковода: 0 - 40-дорожечный дисковод, 1 - 80-дорожечный дисковод. 1-й бит этого числа равен 1, если дисковод двусторонний. 0-й бит этого числа равен 0, если выполняется команда по использованию 80-дорожечного дисковода, как 40-дорожечного.

Начиная с адреса 23773, 8 байт занимает имя файла в кодах ASCII. При инициализации интерфейса в эти ячейки заносится имя "boot". Так осуществляется подготовка к считыванию с диска программы-загрузчика.

Следующий байт после имени файла, ячейка 23781, определяет его тип. Это один из

четырех кодов ASCII - "B", "C", "D" или '#".

Следующие два байта, с адреса 23782, это параметр "START". Если тип файла - "C", то это начальный адрес размещения кодов в памяти Спектрума. Если же тип файла "B", то двухбайтное число в этих ячейках является системной информацией, связанной с автостартом Бейсик-программы.

Два байта с адреса 23784 обозначают длину файла в байтах. Один байт по адресу 23786 определяет длину файла в секторах (по 256 байт). Разумеется, должно иметь место выражение:

```
PEEK 23786 = PEEK 23785 + (PEEK 23784<>0)
```

Очевидно, что если младший байт длины файла имеет хоть какое-то значение, отличное от нуля, то должен быть выделен сектор для размещения файла.

Когда происходит поиск файла на дискете по заданному имени и типу, считывается содержимое нулевой дорожки. В том случае, если такой файл найден, его номер заносится в ячейку 23838. А в ячейки 23787 и 23788 заносятся соответственно начальные сектор и дорожка файла согласно его расположению на диске. Таким образом, теперь имеется вся необходимая информация для загрузки файла в память.

Если Вы выполняете, например, команду:

```
A> LOAD "B:name"
```

то номер дисковода, указанного Вами в качестве временного (в данном случае это "B"), заносится в ячейку 23798. Для дисковода "A" это 0, "B" - 1, "C" - 2, "D" - 3.

Ячейка 23801 содержит признак операции LOAD/VERIFY. Значение 0 соответствует чтению, а 255 - сравнению файла. Вы можете проделать интересный эксперимент: сделайте POKE 23801,255, а затем подайте команду TR-DOS, например, LOAD "name" CODE. Однако вместо загрузки будет выполнена верификация.

Интересна информация, находящаяся в ячейках 23802...23805. Здесь (по одному байту на каждый дисковод) расположен код, определяющий время перемещения головки дисководов A...D. Число, стоящее там после инициализации - 8. Оно соответствует наибольшей скорости перемещения головки. Если чтение сектора прошло неудачно, то это число увеличивается на единицу, то есть становится 9 - это соответствует меньшей скорости перемещения головки. Она отводится на нулевую дорожку, после этого повторяется чтение. Если опять неудача, то опять уменьшение скорости (PEEK 23802=10) и повторяется чтение и так до PEEK 23802=11. Это наименьшая скорость перемещения головки. Вы можете искусственно увеличить ее после неудачного чтения, подав команду (для дисковода "A", например): POKE 23802,8. (Кстати тот же эффект даст POKE 23802,12.)

Вышесказанное про скорость перемещения головки относится к TR-DOS версии 5.01. В более поздних версиях, например, 5.04s, установлена только быстрая скорость перемещения головки и она не зависит от содержимого ячеек 23802...23805. Хотя при инициализации дисковода туда по-прежнему заносится 8. Это число является признаком того, что инициализация произведена (до инициализации каждого из дисководов A...D в этих ячейках содержится значение 255).

В ячейке 23814 задается число байт, по которым в каталоге диска ищется файл. Начальное значение этого параметра - 9 (8 байт - имя файла плюс 1 байт - его тип).

Ячейка 23823 содержит код ошибки TR-DOS. На коде ошибки необходимо остановиться поподробнее. Наберите и выполните следующую Бейсик-программу:

```
10 RANDOMIZE USR 15619 : REM : LOAD "abcde" CODE
20 PRINT PEEK 23823
```

В том случае, если файл "abcde" отсутствует на диске, то сообщения об ошибке не последует: программа остановится с сообщением 0 OK. 20:1. О наличии ошибки свидетельствует только единица на экране. По таблице кодов ошибок (смотри в описании TR-DOS) это соответствует ситуации "нет файла". Код ошибки из Бейсик-программы можно подучить еще и другим способом. Ранее в "ZX-РЕВЮ" мы уже говорили о том, что вместо

RANDOMIZE USR … вполне можно использовать LET S=USR … Все это относится точно также и к адресу 15619. Измените нашу программу:

```
10 LET err = USR 15619 : REM : LOAD "abcde" CODE
20 PRINT err
```

Результат работы этой программы будет точно такой же, как и первой. Можете использовать значение err в Бейсик-программе по своему усмотрению, например:

```
20 IF err<>0 THEN PRINT "ERROR"
```

Продолжаем разговор о системных переменных TR-DOS.

Ячейка 23830 это копия системного регистра. Если посмотреть на схему контроллера дисковода, то это тот регистр, который собран на микросхеме 555ТМ9. Интересная особенность, но у меня пока что нет никаких данных о том, как практически это можно использовать.

Ячейка 23833 определяет номер накопителя A...D (A-0, B-1, C-2, D-3). Можете проделать небольшой эксперимент. Войдите в TR-DOS. Вы увидите приглашение: A>. Теперь выполните: *"b". Приглашение сменится на: B>. Теперь перейдите из TR-DOS в Бейсик, выполнив RETURN, затем сделайте POKE 23833,2 и войдите опять в TR-DOS. Вместо приглашения: B> Вы увидите: C>, так как искусственно поменяли дисковод, изменив содержимое системной ячейки 23833.

Перечислены основные, но далеко не все системные переменные TR-DOS. Если кто-нибудь из читателей обладает более полной информацией, напишите об этом.

Теперь рассмотрим подробнее вопрос о том, как информация хранится на дискете.

ФОРМАТ ЗАПИСИ НА ДИСК.

Данные на дискету записываются в виде секторов, имеющих объем 256 байт. Вдвое меньший, по сравнению с IBM PC объем одного сектора позволяет несколько сэкономить потери из-за недоиспользования места, так как длина файла не кратна объему одного сектора. При большом числе коротких файлов это дает определенный выигрыш.

Предположим, что имеется двусторонний 80-дорожечный дисковод (во всяком случае, таких, наверное, большинство). Если Вы отформатируете дискету, то при завершении форматирования увидите, что на дискете имеется 2544 свободных сектора. Но 80 дорожек умножить на 16 секторов получится 2560 секторов. Где остальные? Дело в том, что нулевая дорожка дискеты 16 секторов - является системной. А файлы могут располагаться начиная с 1-й дорожки по 159 (так как запись двусторонняя).

Информация на нулевую дорожку записывается в строго определенном формате. Эти 16 секторов поделены на две части. В первой части - сектора с 0 по 7 - хранятся заголовки файлов. При этом на один заголовок отводится 16 байт информации. Всего на восьми секторах может быть размещено 128 заголовков файлов. То есть:

8 секторов * 256 байт = 2048 байт

то же самое, что:

126 файлов * 16 байт = 2048 байт.

Из максимального числа файлов, кстати, происходит название: "БЕТА-ДИСК 128".

Для тех, кто программирует в машинных кодах может оказаться полезным еще и такой взгляд. Порядковый номер заголовка имеет следующее свойство: старший полубайт этого числа (старшие 4 бита) соответствует номеру сектора, в котором записан заголовок файла, а младший полубайт (младшие 4 бита) соответствует номеру заголовка в секторе.

О том, как в память компьютера считать нулевую дорожку - позже. А сейчас о формате заголовка файла. Он в точности совпадает с 16 ячейками системных переменных с 23773 по 23788. Из 16 байт, отведенных на один заголовок, первые восемь занимает имя файла в ASCII. Следующий байт - тип файла (тоже в ASCII) - B, C, D или #. Следующие два байта - параметр START, далее 2 байта - длина файла, следующий байт - объем файла в секторах и

далее 1 байт - номер начального сектора файла и 1 байт - номер начального трека (дорожки) файла на дискете.

Кроме сказанного, надо добавить следующее. При удалении файла первый символ имени (первый байт заголовка) заменяется на код 01. В таком виде файл считается удаленным, однако, если Вы не делали команду MOVE, то удаленный файл всегда можно восстановить, изменив первый символ его имени в заголовке либо при помощи программ типа "DISK-DOCTOR", либо при помощи всевозможных программ-оболочек типа "MOA-SERVICE". Однако, если удаляется последний файл каталога, то первый символ имени заменяется не на 1, а на 0. По значению 0 определяется конец каталога. Заголовки вновь записываемых файлов будут располагаться в каталоге, начиная с этого места.

Кроме секторов 0...7, в которых располагаются заголовки файлов, есть еще один сектор, восьмой, в котором содержится общая информация по диску. Эта информация расположена не с начала сектора, а ближе к его концу. Если начать нумерацию байтов сектора с 0, то вот содержание байтов, несущих информацию по диску.

Байты 225 и 226 хранят начало свободного места на диске, доступного для записи файлов - сектор и трек соответственно. Сразу же после форматирования дискеты эти байты имеют значения 0 и 1 соответственно (как говорилось выше, запись файлов на дискету начинается с дорожки номер 1).

Байт 227 определяет тип разметки диска. Значения этого байта могут быть следующими.

22 (#16) - 80 дорожек, 2 стороны;
23 (#17) - 40 дорожек, 2 стороны;
24 (#18) - 80 дорожек, 1 сторона;
25 (#19) - 40 дорожек, 1 сторона.

Байт 228 определяет общее количество файлов, записанных на диск. Это число включает в себя и удаленные файлы, так как физически они все равно присутствуют на диске. То есть это все файлы от начала каталога до начала свободного места.

Байты 229 и 230 (младший и старший) определяют количество свободных секторов на диске. Сразу же после форматирования дискеты командой TR-DOS: FORMAT "name", они равны соответственно 240 и 9. То есть 240+9*256=2544. Если для форматирования воспользоваться программой "DCU" Н. Родионова, можно отформатировать диск на максимальное число дорожек, для большинства дискет практически до 83-84 дорожек. При этом байты 229 и 230 сразу же после форматирования могут иметь другие значения, например 2640 или 2672. На первый взгляд может показаться, что на такую дискету удастся записать больше файлов. Но это только на первый взгляд. Попробуйте отформатировать дискету таким образом, а потом записать на нее файл объемом, например, 1 сектор. После записи Вы увидите, что на дискете осталось 2543 свободных сектора (2544-1 = 2543). То есть TR-DOS пересчитывает число свободных секторов при записи файла исходя не из того, что записано в байтах 229 и 230 системного сектора, а из того, что на диске может быть только 80 или 40 дорожек по 16 секторов.

Единственным преимуществом форматировании на максимальное число дорожек пока что видится возможность использования места за 80-й дорожкой для каких-то своих целей, например, там может располагаться пароль, защищающий дискету от копирования. Практически такой способ защиты очень распространен, но это - тема для отдельного разговора.

Если дискета форматировалась при помощи программы "DCU", то байты 223 и 224 содержат максимальное число форматированных секторов, доступных для записи файлов. Сразу же после форматирования дискеты значения этих байтов совпадают со значением байтов 229 и 230. Но значения последних изменяются по мере записи файлов на дискету, а значения байтов 223 и 224 остаются неизменными и используются при работе программы "DCU".

Если дискета форматировалась командой TR-DOS: FORMAT "name", то содержимое байтов 223 и 224 равно 0. Ненулевое значение этих байтов является признаком того, что

дискета форматирована при помощи программы "DCU".

Байт 231 всегда хранит код 16 (#10). Он определяет число секторов на одной дорожке и никогда не меняет своего значения. Это число обозначает принадлежность диска к операционной системе TR-DOS.

Байт 244 содержит число удаленных файлов - то есть тех, первый символ имени которых в каталоге диска имеет значение 01.

Байты с 245 по 252 хранят имя диска в кодах ASCII (всего - 8 байт).

Сектора с 9 по 15 не несут никакой информации, связанной с обычным функционированием файловой системы диска.

Вот это все, что касается нулевой дорожки. К сказанному можно добавить, что искажение системной информации диска и каталога диска широко используется для зашиты от копирования, и разговор об этом будет уже в этой статье, но чуть позже. А сейчас мы подошли, пожалуй, к самой главной теме.

ВЫПОЛНЕНИЕ КОМАНД TR-DOS ИЗ МАШИННОГО КОДА.

Трудно переоценить значение этих сведений для программиста, поэтому сразу к делу.

В фирменном руководстве по TR-DOS описан способ выполнения команд TR-DOS из машинного кода. Суть его в двух словах в следующем. Где-то в памяти Спектрума размещается фрагмент Бейсик-программы - некоторая фиктивная Бейсик-строка, содержащая команду TR-DOS. Формат записи этой Бейсик-строки такой же, каким он является в обычной Бейсик-программе. Затем в регистры процессора и системные переменные Бейсика заносится адрес, в котором находится новый фрагмент "фиктивной" Бейсик-программы. Затем происходит запуск интерпретатора Бейсика для выполнения этой фиктивной строки. Такой способ выполнения машинного кода при помощи Бейсика настолько громоздок и несуразен, что может иметь разве что чисто теоретическое значение и вряд ли кем-то может восприниматься серьезно. Но главное в том, что он выполняет только то, что может быть выполнено из обычного Бейсика и не более того.

Для выполнения команд из машинного кода в TR-DOS существуют специальные возможности. Основная из них - использование подпрограммы TR-DOS с точкой входа 15635 (#3D13). С этого адреса в конечном счете осуществляется переход к подпрограмме 10300 (#283C), которая в зависимости от кода, содержащегося в регистре C процессора передает управление соответствующей процедуре системы TR-DOS. Практически фрагмент выполнения команды TR-DOS из машинного кода может выглядеть следующим образом:

```
...
LD C, ...
CALL #3D13
...
```

Кроме регистра C может потребоваться предварительная установка других регистров процессора.

Рассмотрим кратко основные процедуры, выполняемые в зависимости от содержимого регистра C.

C=0. Восстановление, сброс контроллера, при котором головка отводится на нулевую дорожку и ожидается появление сигнала INTRQ. Выход из режима ожидания может быть произведен при нажатии клавиши BREAK на клавиатуре.

C=1. Инициализация дисковода. Предварительно в регистре A должен быть задан номер накопителя 0...3. Здесь надо сказать, что если TR-DOS стартует при открытом кармане дисковода или без дискеты, то по адресам 23802...23805 заносятся числа 255. Это означает, что дисководы не инициализированы. В результате работы этой процедуры устанавливается время перемещения головки дисковода - заносится число 8 по одному из адресов 23802...23805 в зависимости от номера дисковода. Далее определяется количество дорожек дисковода и результат заносится по одному из адресов 23752...23755 в

соответствии с номером накопителя (первоначальное значение также 255). Если дисковод имеет 40 дорожек, то записывается 0, если 80 дорожек, то 128 (устанавливается 7-и бит). Далее задается номер дисковода для временной операции (берется из регистра A) и это значение заносятся по адресу 23798. По адресу 23830 помещается копия системного регистра.

Если время перемещения головки (ячейки 23802...23805) уже установлено, то операции по инициализации дисковода не производятся.

C=2. Позиционирование. Предварительно в регистр A помещается номер дорожки, на которую должна быть установлена головка. При двустороннем дисководе первому физическому треку соответствуют номера 0 и 1, второму - 2 и 3 и т. д.

C=3. Эта процедура помещает содержимое регистра A по адресу 23807, задавая, таким способом номер сектора.

C=4. Процедура помещает содержимое регистра HL по адресу 23808 (2 байта), задавая таким способом адрес буфера.

C=5. Очень важная подпрограмма. Она позволяет читать блок секторов. Для этого необходимо подготовить начальные параметры в регистрах процессора. В регистр B помещается количество читаемых подряд секторов. В том случае, если B=0, то с диска считывается только область заголовка, считывание сектора в память не производится. В регистровую пару DE помещается начало считываемого блока секторов: в D - дорожка, а в E - сектор. В регистре HL должен быть задан адрес начала буфера в памяти Спектрума, куда будет производиться загрузка блока секторов.

C=6. Процедура записи на диск блока секторов. Все параметры аналогичны тем, которые были в предыдущей процедуре.

C=7. Вывод каталога диска (аналогично команде CAT TR-DOS). В регистре A должен быть задан канал для вывода каталога. Для вывода на дисплей, например, в регистре A должно быть 2 (кстати, при этом будет очищен экран). Для вывода на принтер - 3. Следует также сказать, что перед выполнением этой подпрограммы можно задать номер накопителя для временной операции, поместив его по адресу 23798. Эта процедура предварительно выполняет процедуру при C=22 - загрузку системного регистра (см. ниже).

C=8. Чтение заголовка файла. В регистре A должен быть номер интересующего файла (с 0 по 127). Из каталога диска 16 байт заголовка файла будут помещены в память, начиная с адреса 23773. Это может быть и удаленный файл, так как не проверяется, есть ли такой файл в действительности.

C=9. Запись в каталог заголовка файла. 16 байт из памяти, начиная с адреса 23773, переписываются в каталог диска на место заголовка файла, номер которого задается в регистре A аналогично предыдущей процедуре.

C=10. Процедура поиска файла. В каталоге диска ищется файл, который по имени и типу совпадает с тем, что задано в памяти, начиная с адреса 23773. Количество байт, на которые ведется поиск, задается по адресу 23814. Начальное значение - 9. В том случае, если файл найден, то его номер будет находиться в регистре C, а также в ячейке 23838. Если файл не найден, то в регистре C будет 128 (7-й бит установлен в 1), содержимое 23838 не изменяется, а в ячейке 23823 будет 255.

C=11. Запись кодового файла. При этом с адреса 23773 задается имя и тип файла, в регистре HL - адрес начала файла в памяти, в регистре DE - длина файла. Эта процедура

выполняет действия аналогичные команде TR-DOS:

```
SAVE "name" CODE Start, Length.
```

C=12. Запись Бейсик-программы. С адреса 23773 должны быть заданы имя и тип файла. В том случае, если тип файла отличается от "B", то файл записывается под именем "boot" <B>.

C=14. Загрузка или верификация файла. Имя и тип файла должны быть помещены по адресу 23773. Здесь есть несколько вариантов, в зависимости от значения регистра A. При A=0 адрес загрузки и длина файла берутся из каталога. При A=3 загрузка происходит с адреса, который задается в регистре HL, причем длина загрузки определяется значением регистра DE. При A=255 адрес загрузки берется из HL, а длина загружаемого файла - из каталога диска. Значение по адресу 23801 определяет команду: 0 - LOAD, 255 - VERIFY.

C=18. Удаление файла. Имя и тип файла должны быть заданы с адреса 23773. При этом удаляются все файлы с такими данными. Как уже говорилось выше, если файл не последний, первый символ имени файла заменяется кодом 01.

C=19. Эта процедура перебрасывает 16 байт информации с адреса, указанного в HL в адрес 23773.

C=20. Процедура, обратная предыдущей. 16 байт перебрасываются с адреса 23773 в адрес, указанный в HL.

C=22. Процедура загружает системный регистр интерфейса. Код - в аккумулятор (регистре A), при загрузке к содержимому аккумулятора добавляется #3C.

Для других значений C в различных источниках приводятся противоречивые или весьма смутные сведения, поэтому на них не будем останавливаться. Для любителей покопаться в ПЗУ TR-DOS привожу входные адреса процедур, соответствующие различным значениям регистра C.

C=0	#3D98	C=13	#01D3
C=1	#3DCB	C=14	#290F
C=2	#3E63	C=15	#01D3
C=3	#3F02	C=16	#01D3
C=4	#3F06	C=17	#01D3
C=5	#1E3D	C=18	#2926
C=6	#1E4D	C=19	#28E0
C=7	#28D8	C=20	#28E3
C=8	#165C	C=21	#2739
C=9	#1664	C=22	#1FEB
C=10	#1CF0	C=23	#1FF6
C=11	#28FB	C=24	#0405
C=12	#28F2		

Для выполнения команд TR-DOS из машинного кода существует еще один способ. Он позволяет обращаться сразу же непосредственно в теневое ПЗУ TR-DOS, в любую его точку. Вы спросите, как же это практически сделать, ведь обратившись, например, по одному из тех адресов, которые приведены выше, мы попадем всего лишь в основное ПЗУ Спектрума. Для этого в системе TR-DOS имеется специальная точка входа с адресом 15663 (#3D2F). Там находится всего лишь одна команда NOP, после чего сразу же идет RET. Дело в том, что при обращении по адресу 15663 селектор адреса БЕТА-ДИСК интерфейса подключает

теневое ПЗУ, а выполнение команды RET приведет к переходу программы на адрес, записанный на стеке. Поэтому если перед обращением к 15663 на стек поместить нужный адрес, то можно осуществить переход на этот заданный адрес теневого ПЗУ командой: JP 15663.

В качестве практического примера использования машинного кода для выполнения команд TR-DOS приведу простую программу "DIR", которая позволяет выводить на экран параметры файлов (имя, тип, адрес загрузки, длина и т. п.), содержащиеся в каталоге диска. Кроме теоретического, эта программа представляет определенный практический интерес, так как выводит кроме обычной информации еще и данные о начале файлов на диске (команда LIST TR-DOS эту информацию не выводит).

Программа "DIR".

Машиннокодовая часть программы выглядит следующим образом (адрес размещения - произвольный):

```
21 00 AB        LD    HL,#AB00    (1)
11 00 00        LD    DE,#0000    (2)
06 08           LD    B,#08       (3)
0E 05           LD    C,#05       (4)
CD 13 3D        CALL  #3D13       (5)
C9              RET               (6)
```

В регистре HL задается (1) адрес буфера, куда будет производиться чтение каталога диска (#AB00=43776). В регистрах D и E задаются начальные дорожка и сектор (2): чтение будет начинаться с 0 сектора 0 дорожки. Число считываемых подряд секторов задается (3) равным 8, столько занимает каталог диска. Затем задается (4) параметр, определяющий процедуру: C=5, то есть чтение блока секторов и далее (5) выполнение подпрограммы TR-DOS. После чего (6) - возврат в вызывающую программу.

После того, как содержимое нулевой дорожки считано в буфер, его можно рассмотреть более подробно. Этим уже займется Бейсик. Программа выглядит следующим образом:

```
2 BORDER 1: PAPER 7: INK 1: CLEAR 39999
3 RESTORE : FOR a=40000 TO 40013: READ b: POKE a, b: NEXT a: DATA 33,0,171,17,0,0,6,8,14,5,
      205,19,61,201
4 GO TO 100
5 RANDOMIZE USR 15619: REM : SAVE "DIR"
6 STOP
20 PRINT AT 0,0; PAPER 6; INK 2;"Name "; PAPER 2; INK 7; "Type" ; PAPER 3; INK 0; " Start
      Length " ; PAPER 4; "Beginn"; PAPER 5;" Len"
22 PRINT BRIGHT 1: INVERSE 1; " ASCII "; INVERSE 0; " B y t e s "; INVERSE 1;"Trk"; INVERSE
      0; " Sector"
24 RETURN
30 FOR a=a TO a+7: LET n$=n$+CHR$ PEEK a: NEXT a: REM name
31 LET t$ = CHR$ PEEK a: REM type
32 LET a=a+1: LET st=PEEK a+256*PEEK (a+1): REM start
33 LET a=a+2: LET len=PEEK a+256*PEEK (a+1): REM length
34 LET a=a+2: LET q=PEEK a: REM length SEC
35 LET a=a+1: LET sec=PEEK a: REM benin SEC
36 LET a=a+1: LET tr=PEEK a: REM begin TRK
37 LET a=a+1: RETURN
100 RANDOMIZE USR 40000
200 LET a=43776
300 GO SUB 20: LET j=2: BEEP .01, j+10
1000 LET n$="": GO SUB 30
1010 PRINT AT i,j; PAPER 6; INK 2; n$; PAPER 2; INK 7; t$;
1020 PRINT PAPER 3; INK 0; TAB 10; st; TAB 16; len; TAB 22; PAPER 4; INK 0; TAB
      (22+(tr<100)); tr; TAB 26; sec; TAB 28; PAPER 5; INK 0; TAB 29; q; TAB 32;
1030 LET j=j+1: BEEP .01,.j+10: IF J>=22 THEN PAUSE 0: CLS : GO TO 300
1040 IF a<43776+256*8 THEN GO TO 1000
2000 PRINT #0; INVERSE 1; "DIR O.K.": BEEP .1,26: BEEP .1,20: PAUSE 0: CLS : GO TO 200
```

После набора программы сделайте RUN 5 - это запись программы на диск. Нет необходимости записывать программу на диск с автостартом. Любой загрузчик ("boot") запускает программу с начальной строки. При этом строка 2 формирует в памяти необходимый кодовый блок, после чего происходит чтение восьми секторов нулевой дорожки (строка 100). Считанная информация размещается с адреса 43776 (#AB00). GO SUB 20 в строке 300 - это распечатка "шапки". Переменная j - это номер строки экрана для вывода последующей информации.

Со строки 1000 расположена программа распечатки одного экрана информации. Подпрограмма GO SUB 30 производит расчет параметров для каждого заголовка файла. Первые 8 байт - имя программы (n$, строка 30). Затем тип (t$, строка 31). Следующие 2 байта определяют параметр START (st, строка 32). Затем 2 байта - длина файла (len, строка 33). Потом идет 1 байт - длина файла в секторах (q, строка 34). Следующая пара байт начало файла на диске - сектор и дорожка (соответственно sec и tr в строках 35 и 36).

Строки 1010 и 1020 - это распечатка полученных данный по одному заголовку - 16 байтам нулевой дорожки. Строка 1030 - завершающие операции по распечатке заголовка и проверка на достижение конца экрана. Если он достигнут, то очистка экрана и распечатка следующего (со строки 300).

Строка 1040 проверяет, не достигнут ли конец каталога (8 секторов по 256 байт). Если нет, то переход к расчету и печати информации о следующем файле (со строки 1000). Иначе - завершение работы переходом на строку 2000, где программа зацикливается, возобновляя вывод информации с первого заголовка файла.

Теперь предположим, что Вы запустили эту программу, вставив в дисковод какую-нибудь дискету. После того, как на экран будет выведена информация о всех существующих файлах (при этом удаленные файлы будут начинаться со знака "?"), если на дискете больше нет файлов, то вся последующая часть каталога будет состоять из нулей. Чтобы не "прокручивать" вхолостую весь каталог, можно добавить в программу такую строку:

```
1035 IF PEEK a=0 THEN GO TO 2000
```

Так как начало свободного места на диске обозначается нулевым первым кодом имени файла, то как только встретится такой код, вывод будет завершен переходом на 2000 строку.

Как еще можно усовершенствовать программу? Ну, например, можно в конце добавить вывод информации по диску, его имя, число файлов, наличие свободного места и т.д., в общем, все то, что можно "выудить" из восьмого системного сектора. Для этого надо чтобы с нулевой дорожки было считано не 8 секторов, а 9. Для этого в строке 3 после DATA число 8 должно быть заменено на 9. Теперь заодно с каталогом в память будет считан и системный сектор. Попробуйте сами сделать такое усовершенствование, если хотите, расположив эту часть программы, начиная со строки 2000. Вся необходимая информация по расшифровке системного сектора приведена выше.

Для того, чтобы просмотреть новый диск, можно нажать BREAK, затем сделать RUN. Используя блок кодов "ON ERROR GO TO" (см. ZX-РЕВЮ N 5-6 за 1992 год, стр.113), Вы можете еще более усовершенствовать программу: она будет перезапускаться с начальной строки при нажатии BREAK для того, чтобы можно было просмотреть новый диск.

ПРИЕМЫ ЗАЩИТЫ ОТ КОПИРОВАНИЯ.

До сих пор мы вели разговор о том, как все должно работать, как говорится, "по букварю". Теперь подойдем более критически к некоторым вопросам. Например, длина файла. В 16 байтах заголовка длина файла фигурирует дважды. Действительно, длина файла это два байта: 11-й и 12-й. Кроме того, это 13-й байт - длина файла в секторах. Зачем он нужен? Ведь он же легко рассчитывается из известных 11 и 12 байтов. TR-DOS устроена так, что точная длина файла (11-12 байты) необходима для процедуры загрузки файла в память, а вот при операциях перезаписи файла TR-DOS оперирует именно с 13 байтом,

резервируя на диске столько места, сколько обозначено в 13 байте. Поэтому если искусственно, "вручную", например, при помощи программы типа "DISK-DOCTOR", изменить значение этого байта, задав нулевое значение, то файл будет загружаться и стартовать нормально (11 и 12 байты не изменены), но при перезаписи файла на другую дискету будет переписано ноль секторов, то есть фактически перезаписи не произойдет.

На этом принципе работают некоторые самые простые программы по защите файлов от копирования, например программа "CLOSE". Эта программа обнуляет 13-й байт заголовка у файлов, имеющих расширение "B", задавая, таким образом, нулевую длину файла в секторах. Кроме того, эта программа искажает системную информацию по диску, изменяя значения параметров системного сектора. Искажению подвергаются: начало и количество свободного места на диске, тип диска (вместо двустороннего задается односторонний), количество файлов. Кроме того, в имя первого файла вводятся непечатаемые управляющие коды. Однако при этом не нарушается нормальный старт программ с диска. Такие диски удается скопировать только целиком при помощи специальных копировщиков типа "диск в диск", когда копируется без изменения содержимое всех 80 дорожек диска. Кстати, изложенные в этой статье сведения позволяют Вам самостоятельно разработать такой копировщик даже на Бейсике, с выполнением одного фрагмента в машинных кодах, аналогично программе "DIR". Пофайловое копирование всего диска при помощи команды TR-DOS: COPY B, а также копирование при помощи программ-оболочек типа "MOA-SERVICE" в этом случае не помогает.

Этот способ защиты широко распространен на рынке программного обеспечения, но пользуются им не разработчики оригинальных программ, а скорее "работники торговли". Их дискеты недостаточно хорошо скомпонованы, чтобы копировать их целиком. Обычно на дискете бывает несколько хороших программ, остальное - мусор для заполнения дискеты. Впрочем, это уже отклонение от темы. Так или иначе, Вы должны быть свободны в вопросах компоновки своих дискет, поэтому я поделюсь с Вами программой, созданной мной в противовес "CLOSE". Она называется "OPEN". На самом деле проблема достаточно примитивна. Испорченную длину файла в секторах легко можно восстановить, вычислив разность между началом этого файла (байты 14 и 15 заголовка) и началом следующего файла. Заодно программа "OPEN" также рассчитывает объем и начало свободного места на диске и восстанавливает всю остальную системную информацию по диску.

Программа "OPEN".

```
2 BORDER 7: PAPER 7: INK 0: CLEAR 39999
3 RESTORE : FOR a=40000 TO 40012: READ b: POKE a,b: NEXT a:DATA 33,0,171,17,0,0,1,5,9,
      205,19,61,201
4 GO TO 100
5 RANDOMIZE USR 15619: REM : SAVE "OPEN"
6 STOP
100 CLS : PRINT #0; "INSERT DISK AND PRESS ANY KEY": PAUSE 0
110 INPUT INKEY$: PRINT #0;TAB 8; FLASH 1;" PLEASE WAIT "
200 RANDOMIZE USR 40000
210 LET a=43776: LET n=0: LET del=0: LET free=2544
220 GO SUB 1000
230 LET a=43776+128*16
240 GO SUB 2000
300 INPUT INKEY$: PRINT #0: "OWERWRITE TRACK <0> (Y/N) ? ": PAUSE 0: INPUT INKEY$
310 IF INKEY$="y" OR INKEY$="Y" THEN POKE 40007,6: RANDOMIZE USR 40000
320 STOP
1000 REM Katalog
1010 IF PEEK a=0 THEN RETURN
1020 LET n=n+1
1030 IF PEEK a=1 THEN LET del=del+1: GO TO 1050
1040 IF PEEK a<32 OR PEEK a>127 THEN POKE a,63
1050 LET beg1=PEEK (a+14)+16*PEEK (a+15)
1060 LET beg2=PEEK (a+14+16)+16*PEEK (a+15+16)
1070 LET lens=beg2-beg1
1080 IF PEEK (a+16)=0 THEN LET lens=PEEK (a+12)+(PEEK (a+11)<>0)
1090 POKE (a+13), lens
```

```
1100 LET free=free-lens
1110 LET a=a+16
1120 GO TO 1000
2000 REM Disk-data
2010 LET space=beg1+lens
2020 LET spt=INT (space/16): LET sps=space-spt*16
2030 POKE (a+225),sps: POKE (a+226),spt
2040 POKE (a+227),22
2050 POKE (a+228),n
2060 LET fr2=INT (free/256): LET fr1=free-fr2*256
2070 POKE (a+229),fr1: POKE (a+230),fr2
2080 POKE (a+244), del
2090 RETURN
```

По своей структуре программа похожа на "DIR". После старта с начальной строки, программа формирует кодовый блок для чтения девяти секторов нулевой дорожки, затем останавливается (строка 100) для того, чтобы можно было вставить в дисковод засекреченный диск. После нажатия клавиши происходит чтение нулевой дорожки, затем начинаются расчеты. Задаются начальный адрес каталога (a), число файлов (n) и число удаленных файлов (del) принимают для начала нулевые значения, а число свободных секторов на диске (free) - максимальное значение для двусторонней дискеты 80 дорожек - 2544 сектора.

Подпрограмма со строки, 1000 приводит в порядок каталог диска. Строка 1010 проверяет, не достигнуто ли свободное место на диске и если да, то выход из подпрограммы. Если нет, то число файлов увеличивается на единицу (строка 1020), затем проверяется, не удаленный ли это файл (строка 1030). Если да, то счетчик удаленных файлов увеличивается на единицу строка 1040 исправляет значения тех кодов в названии файла, которые могли быть искусственно изменены для невозможности произвести распечатку каталога командами TR-DOS: CAT или LIST. Управляющие коды либо токены ключевых слов, подставленные в имя файла заменяются символом ASCII с кодом 63 - это знак вопроса.

Далее рассчитывается длина файла в секторах (lens). Для этого определяется начало текущего файла (beg1), затем начало следующего файла (beg2). Эти значения рассчитываются непосредственно в секторах от начала диска. Их разность (строка 1070) определяет длину текущего файла. Однако в том случае, если текущий файл является последним на диске (строка 1080), то значение beg2 даст нулевое значение. Поэтому длину файла в секторах в этом случае можно рассчитать непосредственно по его длине в байтах. После занесения рассчитанного значения в память (строка 1090), рассчитывается оставшееся свободное место на диске (строка 1100), затем - переход к следующему заголовку путем возврата на строку 1000.

После того, как выполнена подпрограмма восстановления каталога (в строке 220), значение адреса устанавливается на начало системного сектора (строка 230). Подпрограмма со строки 2000 восстанавливает системную информацию диска. Начало свободного места на диске в секторах (space) определяется как начало последнего файла плюс его длина (строка 2010). Далее это значение преобразуется (строка 2020) в номер дорожки (spt) и сектора (sps) и заносится в память (строка 2030).

Строка 2040 устанавливает тип диска: двусторонний, 80 дорожек. Далее в память заносится общее число файлов (строка 2050), рассчитываются старший и младший байты числа свободных секторов (строка 2060) и полученные значения заносятся в память (строка 2070). Заносится в память и число удаленных файлов (строка 2080). Далее - возврат из подпрограммы на строку 300. Здесь выводится предупредительная табличка и, в случае подтверждения (клавиша "Y"), происходит перезапись нулевой дорожки диска (строка 310). Здесь значение регистра C в ячейке 40007 вместо чтения (C=5) заменяется на запись (C=6). После этого программа останавливается. Можете запускать дискету обычным порядком и копировать файлы с нее на другие дискеты.

FORUM

Русификация резидентной процедурой

Тема русификации это, пожалуй, вечная тема для "Спектрума". Пока что не существует способ (и будет ли когда-нибудь существовать), который бы был достаточно универсален и хорош для всех. Многие программисты ломают голову над решением этой проблемы. В этот раз интересное письмо пришло к нам от Александра Еременко из Свердловска. Он прислал программу в машинных кодах, позволяющую переключать символьный набор одновременным нажатием двух клавиш (SYMBOL SHIFT)+(ENTER). При первом нажатии включается загруженный в ОЗУ символьный набор, а при повторном нажатии - опять символьный набор из ПЗУ. Самое главное, что такое переключение можно сделать в любой момент как пишет Александр, неважно, где Вы работаете, в режиме редактирования Бейсик-строк или в программе, все равно можно производить переключение шрифта.

Нам понравилась идея уважаемого корреспондента и мы даем ее всем читателям, снабдив собственными комментариями и постаравшись немного ее развить.

Вот эта программа:

```
CD7C  3ECB      LD    A,#CB
CD7E  2120CD    LD    HL,#CD20
CD81  36CD      LD    (HL),#CD
CD83  2B        DEC   HL
CD84  BC        CP    H
CD85  20FA      JR    NZ,#CD81
CD87  3ECC      LD    A,#CC
CD89  ED47      LD    I,A
CD8B  F3        DI
CD8C  ED5E      IM    2
CD8E  FB        EI
CD8F  C9        RET
CD90  C5        PUSH  BC
CD91  D5        PUSH  DE
CD92  E5        PUSH  HL
CD93  F5        PUSH  AF
CD94  3E7F      LD    A,#7F
CD96  DBFE      IN    A,(#FE)
CD98  FEFD      CP    #FD
CD9A  202A      JR    NZ,#CDC6
CD9C  3EBF      LD    A,#BF
CD9E  DBFE      IN    A,(#FE)
CDA0  FEFE      CP    #FE
CDA2  2022      JR    NZ,#CDC6
CDA4  3A10CD    LD    A,(#CD10)
CDA7  3C        INC   A
CDA8  3210CD    LD    (#CD10),A
CDAB  CB47      BIT   0,A
CDAD  2805      JR    Z,#CDB4
CDAF  01003C    LD    BC,#3C00
CDB2  1803      JR    #CDB7
CDB4  01D0CC    LD    BC,#CCD0
CDB7  21365C    LD    HL,#5C36
CDBA  71        LD    (HL),C
CDBB  23        INC   HL
CDBC  70        LD    (HL),B
CDBD  219701    LD    HL,#0197
CDC0  116400    LD    DE,#0064
CDC3  CDB503    CALL  #03B5
CDC6  F1        POP   AF
CDC7  E1        POP   HL
```

```
CDC8   D1          POP    DE
CDC9   C1          POP    BC
CDCA   C33800      JP     #0038
CDCD   C390CD      JP     #CD90
CDD0   ......      символьный набор
```

Как видим, принцип действия этой программы основан на использовании прерываний второго рода IM 2. Подробнее о прерываниях Вы можете прочитать в нашей книге, посвященной программированию в машинных кодах. Там мы рассказывали, что на использовании прерывания первого рода основан основной режим работы "Спектрума". В этом режиме (если он разрешен командой EI) всегда в ответ на прерывание, поступающее по шине INT (а это происходит 50 раз в секунду), выполняется обращение по адресу 0038H. Процедура, расположенная с этого адреса, обеспечивает сканирование клавиатуры в поисках нажатой клавиши.

Теперь о прерывании второго рода. В компьютерах иных систем, собранных на базе процессора Z-80, для программиста существует возможность самому задать адрес, по которому будет происходить обработка этого прерывания. Он определяется следующим образом. Старший байт поступает из регистра I процессора (он так и называется "вектор прерываний"), а младший байт должно выдавать периферийное устройство, от которого получено прерывание. В паре ячеек, адрес которых определен таким образом, хранится адрес процедуры, обслуживающей прерывание. Программ, обслуживающих такое прерывание может быть 128, так как старший байт определен регистром I, а младший байт - это 256 ячеек, в которых можно записать 128 двухбайтных адресов. В "Спектруме" режим прерываний второго рода не используется для взаимодействия с периферийными устройствами, поэтому появляется возможность использовать этот режим в своих целях.

Теперь конкретно о предложенной программе. Вначале выполняется инициализирующая процедура. Она запускается с адреса загрузки, то есть CD7CH (52604). При этом происходит следующее. Область, начиная с CC00H, и кончая адресом CD20H заполняется кодом CD. Затем в регистре I процессора задается старший байт адреса ячейки памяти, в которой хранится адрес обслуживающей процедуры CC. Далее происходит включение режима прерываний второго рода и выполняется возврат в вызывающую программу.

Что же происходит при получении сигнала на прерывание, (который, как мы помним, приходит 50 раз в секунду) Несмотря на то что неопределен младший байт ячейки с адресом обслуживающей программы, обратившись по любому адресу со старшим байтом CC (то есть начиная CC00H и кончая CCFFH), получим один и тот же адрес обслуживавшей программы CDCDH, так как вся эта область заполнена кодом CD.

По адресу CDCDH (см. листинг программы) видим безусловный переход на адрес CD90H. Это и есть начало программы, обслуживающей полученное прерывание. Перед началом выполнения каких либо действий, на стеке запоминается содержимое всех регистров процессора, которые могут быть изменены при работе обслуживающей процедуры. Затем выполняется сканирование двух полурядов клавиатуры (о том, как производится опрос клавиш, Вы также можете прочитать в нашей книге, посвященной программированию в машинных кодах). Вначале проверяется нажатие клавиши SYMBOL SHIFT и, если она не нажата, то сразу переход на завершение процедуры обработки прерывания. Если SYMBOL SHIFT нажата, то проверяется факт нажатия ENTER. Если нет, то также переход на завершение процедуры, а если и эта клавиша нажата, то происходит дальнейшая работа.

Проверяется содержимое ячейки, определяющей включение одного или другого символьного набора это ячейка CD10H. От значения младшего бита этой ячейки (четное число или нечетное) зависит выбор одного из двух символьных наборов, при этом происходит увеличение на единицу содержимого этой ячейки (то есть подготовка к включению в следующий раз противоположного символьного набора). Соответствующее значение системной переменной CHARS пересылается при помощи регистра BC, в область системных переменных, в ячейку 5C36H. Далее происходит вызов подпрограммы BEEPER из

ПЗУ (параметры: тон и длительность звукового сигнала задаются в регистрах HL и DE). Звуковой сигнал свидетельствует о произошедшем переключении символьного набора.

Далее выполняется финишная процедура: восстановление со стека прежних значений регистров процессора и безусловный, переход на адрес 0038H. Как это должно быть в случае обработки прерываний первого рода, с этого места работа компьютера совпадает с традиционной.

При реализации этого метода следует помнить, что символьный набор располагается сразу же за самой программой, с адреса CDD0H (52688). При этом системная переменная CHARS равна CCD0H. Но Вы можете расположить символьной набор там, где это Вам удобно, изменив значение CHARS в ячейках CDB5H и CDB6H. Кроме того, оба переключаемых символьных набора могут находиться в ОЗУ, и Вы можете поочередно переводить на любой из них. Для этого значение системной переменной chars для второго символьного набора должно быть задано в ячейках CDB0H и CDB1H. Следует также помнить, что для работы программы необходима область кодов с адреса CC00H (52224), то есть перед стартом инициализирующей процедуры (RANDOMIZE U5R 52604) надо зарезервировать это место, изменив RAMTOP командой CLEAR 52223.

Следует также учитывать тот факт, что раз уж мы работаем с прерываниями второго рода, призванными обслуживать периферийные устройства, то работа с последними, например с дисководом, после инициализации блока кодов будет невозможна. Для работы с периферией необходимо восстановить традиционный порядок работы. Об этом еще будет разговор ниже.

Вообще нам очень понравилась идея, заложенная в этой программе и тот изящный способ, при помощи которого эта идея реализуется. Для того, чтобы поглубже "прочувствовать" действие этой программы, наберите простую демонстрационную программку на Бейсике:

```
1 GO TO 100
2 CLEAR 52223: LOAD "rus"CODE 52604: LOAD "cbr" CODE 52688
3 RANDOMIZE USR 52604
4 RUN
100 PRINT "Press <S.S.>+<ENTER> for control"'': PAUSE 0
200 FOR a=32 TO 127; PRINT CHR$ a;: PAUSE 1: NEXT a: GO TO 200
```

После набора запустите её с первой строки командой RUN и внимательно понаблюдайте за реакцией компьютера. Хотя ничего неожиданного сейчас произойти не должно (компьютер не подпрыгнет, он по-прежнему останется на своем месте), но все-таки остановимся на происходящем подробнее. Сразу после старта на экране появится надпись: "нажмите <S.S.>+<ENTER> для управления" и программа остановится на PAUSE 0, ожидая нажатия клавиши. Попробуйте нажать указанную комбинацию клавиш. Пауза прервется (SYMBOL SHIFT здесь ни при чем, сработало нажатие ENTER) и начнется распечатка символьного набора из ПЗУ. Когда заполнится весь экран, появится привычный запрос: "scroll?". Попробуйте опять нажать SYMBOL SHIFT + ENTER. Вывод символов на экран будет продолжен.

Теперь остановите программу нажатием BREAK и запустите ее со второй строки (с которой должен происходить автостарт) RUN 2. При этом загрузите (заготовленные за ранее) исследуемый блок кодов "rus" и любой имеющийся у Вас символьный набор "chr" (лучше "утолщенный" или стилизованный, так будет нагляднее). После загрузки произойдет инициализация программы в кодах, после чего Бейсик программа запустится с начала, как это делали Вы командой RUN. Сравните реакцию компьютера. После появления надписи в верхней строке, нажмите SYMBOL SHIFT + ENTER. Вы услышите короткий звуковой сигнал, свидетельствующий о нажатии на клавиши, но больше ничего на экране не изменилось: пауза не прервалась, как раньше. То есть, указанная комбинация клавиш теперь не влияет никаким образом на работу Бейсик-программы. Теперь она воспринимается именно как SYMBOL SHIFT + ENTER, а не как просто ENTER. Прервите паузу нажатием на любую клавишу, и Вы увидите, что на экран выводится новый символьный набор, то есть

произошло переключение. Пока экран заполняется, можете попробовать несколько раз нажать SYMBOL SHIFT + ENTER и убедиться в том, что происходит переключение символьных наборов. Когда экран будет заполнен и появится запрос "scroll?", опять нажмите комбинацию клавиш. Кроме звукового сигнала, свидетельствующего о переключении символьных наборов, ничего не происходит. Для продолжения работы надо нажать, например, просто ENTER. Можно, таким образом, сделать вывод: приведенный блок кодов не оказывает никакого вмешательства в работу интерпретатора Бейсика, он только исправно выполняет то, что ему предписано - переключает символьные наборы.

Продолжаем наше исследование. Вызовите на редактирование 100 строку Бейсик-программы. Попробуем в кавычках напечатать текст, состоящий из символов разных символьных наборов. Для этого, находясь в середине текста, нажмите SYMBOL SHIFT + ENTER. Звуковой сигнал - переключение произошло, но на экране ничего не меняется до того момента, пока Вы не нажмете какую-нибудь клавишу. А происходит следующее. Весь текст редактируемой строки перерисовывается заново, причем символами нового символьного набора, отсюда второй вывод: нельзя при помощи этого блока кодов решить задачу совмещения символов разный символьных наборов внутри одного оператора PRINT, действительно, чудес ведь не бывает. Кодовый блок только переключает символьный набор, а текст, подлежащий выводу на экран, никак не изменяется, то есть каким было кодовое выражение строки текста, таким оно и осталось, таким и выводится на экран, только в новом символьном наборе. То же относится и к оператору INPUT. Нельзя при вводе текстовых выражений совместить русские и английские буквы.

И еще один вывод. Переключение символьного набора таким способом происходит только "вручную", то есть, если по алгоритму выполнения программы должно произойти переключение символьного набора, то оно должно быть выполнено, как обычно, изменением CHARS при помощи POKE. Так что же, новый изящный метод переключения шрифтов не вносит ничего нового по существу проблемы? Видимо, такое заключение преждевременно. Существуют ситуации, где изложенный способ сможет оказать действительно неоценимую пользу. Например, программа "МОЗАИКА", которая опубликована в этом выпуске ZX-РЕВЮ на с. 1. Там переключение из текстового в графический режим происходит включением регистра CAPS LOCK, из-за чего существует ограниченное число псевдографических символов, только по числу малых букв латинского символьного набора - 26. Если переключать символьные наборы новым способом, то во сколько раз можно расширить графические возможности, а следовательно и привлекательность этой программы! Ведь все до одного 96 символов нового символьного набора можно отдать графическим образам, да и захватить еще часть первого символьного набора, как это сделано в "МОЗАИКЕ".

Другой пример. Вы набираете программу, в которой заведомо будет только русский текст (какой-нибудь тест или обучающая программа). И естественно, хотите для придания программе большей привлекательности, использовать русские заглавные и строчные буквы, используя загружаемый только русский символьный набор. При этом, если Вы набиваете текст программы, используя стандартный символьный набор ПЗУ, то легко можете допустить ошибки при наборе русского текста в операторах PRINT, так как набор ведется практически "вслепую". Если же Вы переключитесь на русский символьный набор, то сама Бейсик-программа, токены, имена переменных - все это станет "нечитаемым". Применение изложенного в этой статье приема позволит вам при наборе и отладке программы без труда переключаться с одного на другой символьные наборы в зависимости от того, что Вы в данный момент предпочитаете видеть.

Мы уверены, что если начать развивать эту тему, то появятся новые оригинальные программы с использованием этого метода. Определенные проблемы для начинающих может вызвать тот факт, что блок кодов нельзя переместить в другое место памяти, но при желании можно это сделать. Мы попробовали поэкспериментировать и у нас получился несколько измененный и немного усовершенствованный вариант этой программы. Вот то,

что у нас получилось.

```
FBF4  3EFB      LD    A,#FB
FBF6  ED47      LD    I,A
FBF8  ED5E      IM    2
FBFA  C9        RET
FBFB  00        NOP
FBFC  ED56      IM    1
FBFE  C9        RET
FBFF  01FC      DEFB  #FC01
FC01  C5        PUSH  BC
FC02  D5        PUSH  DE
FC03  E5        PUSH  HL
FC04  F5        PUSH  AF
FC05  3E7F      LD    A,#7F
FC07  DBFE      IN    A,(#FE)
FC09  FEFD      CP    #FD
FC0B  202D      JR    NZ,#FC3A
FC0D  3EBF      LD    A,#BF
FC0F  DBFE      IN    A,(#FE)
FC11  FEFE      CP    #FE
FC13  2025      JR    NZ,#FC3A
FC15  2132FC    LD    HL,#FC32
FC18  CB7E      BIT   7,(HL)
FC1A  2807      JR    Z,#FC23
FC1C  11003C    LD    DE,#3C00
FC1F  CBBE      RES   7,(HL)
FC21  1805      JR    #FC28
FC23  1158FB    LD    DE,#FB58
FC26  CBFE      SET   7,(HL)
FC28  21365C    LD    HL,#5C36
FC2B  73        LD    (HL),E
FC2C  23        INC   HL
FC2D  72        LD    (HL),D
FC2E  210002    LD    HL,#0200
FC31  117F00    LD    DE,#007F
FC34  CDB503    CALL  #03B5
FC37  00        NOP
FC38  00        NOP
FC39  00        NOP
FC3A  F1        POP   AF
FC3B  E1        POP   HL
FC3C  D1        POP   DE
FC3D  C1        POP   BC
FC3E  C33800    JP    #0036
```

Прежде чем начать пояснения, рассмотрим подробно карту памяти, тех областей, которые имеют значение для работы программы.

```
                        RAMTOP 64499
---------------------------------
Программа в машинных          64500
кодах (новый вариант)         FBF4H
                     77 байт
---------------------------------
Свободное место; для ра-      64577
боты программы не исполь      FC41H
зуется               23 байта
---------------------------------
Символьный набор              64600
(любой по вкусу)              FC58H
                     768 байт
---------------------------------
UDG-графика, стандартно       65368
расположенная здесь           FF58H
                     168 байт
---------------------------------
                              P_RAMT
```

Эта область памяти должна быть зарезервирована при старте Бейсик-программы, путем переустановки RAMTOP командой CLEAR 64499. Файл, содержащий блок кодов, начинается с адреса 64500 и включает в себя, в зависимости от конкретных задач, либо только программу в кодах (до адреса 64577), либо программу и символьный набор (заодно с 23-мя неиспользуемыми байтами), либо еще и блок UDG-графики (то есть по адрес 65535 включительно). Область памяти с адреса 64577 (23 байта) в работе программы не участвует, Вы можете использовать ее по своему усмотрению.

Инициализация выполняется командой RANDOMIZE USR 64500. С этого адреса (FBF4H) начинается инициализируемая процедура. Кстати сказать, нет необходимости заполнять почти 300 байт кодом для задания адреса обслуживавшей процедуры. Дело в том, что при существующей архитектуре "Спектрума" младший байт всегда будет равен FF. То есть, если в регистре I процессора задать, например FB, то при прерывании второго рода переход произойдет всегда на программу, адрес которой хранится в FBFFH. В этой паре ячеек (FBFF - младший байт, в FC00 - старший байт) находится адрес процедуры, обслуживающей прерывание второго рода. Читаем этот адрес (см. листинг программы): FC01.

Начало процедуры, расположенной по этому адресу точно такое же, как и в неполном варианте. Так же опрашивается нажатие интересующих клавишей, а дальше начинается отличие. Системной ячейкой, определяющей, какой символьный набор должен включиться, выбрана ячейка FC32H. Точнее, определяющим является 7 бит числа, находящегося там, (получается либо 7F, либо FF). В этой ячейке содержится параметр, задающий время звукового сигнала для подпрограммы BEEPER, вызываемой из ПЗУ. Таким образом, переключение на символьный набор из ОЗУ сопровождается длинным звуком, а обратное переключение на символьный набор из ОЗУ - коротким.

После вызова подпрограммы BEEPER, зарезервировано 3 свободных ячейки памяти. Здесь, при необходимости, можно организовать вызов каких-то дополнительных сервисных процедур, если это потребуется в будущем (вызов какой-нибудь подпрограммы: CALL ... или переход - JP ... - все это занимает 3 байта) далее - финишные операции и переход на адрес 0033H.

Кроме указанных отличий, новый вариант программы предусматривает возможность отключения режима прерываний второго рода, это необходимо делать прежде, чем, например, обращаться к дисководу. Иначе произойдет сбой, зависание или сброс компьютера. "Де-инициализация" или восстановление традиционного режима работы выполняется командой RANDOMIZE USR 64508. С этого адреса (FBFCH) расположена восстанавливавшая процедура. Здесь включается режим прерываний первого рода, восстанавливая исходный режим работы "Спектрума".

В заключение хочется поблагодарить Александра Еременко за присланную программу. Надеемся, что она подтолкнет и других читателей к решению насущных проблем оригинальными, нетрадиционными методами.

Расчет резонансных характеристик.

Кроме оригинальной программы для переключения символьных наборов, в своем письме Александр Еременко из Свердловска делится с читателями также программой на Бейсике, которую можно использовать для расчета резонансных характеристик пассивных фильтров в диапазоне звуковых частот. Расчет производится для двух типов фильтров по схемам, приведенным на рисунках 1 и 2. Мы приводим эту программу с некоторыми непринципиальными изменениями. Кроме того, мы русифицировали ее по методике, приведенной в "ZX-РЕВЮ"-92г. №1,2, стр. 31.

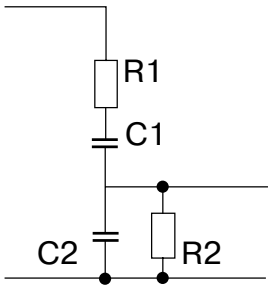


Рис. 1

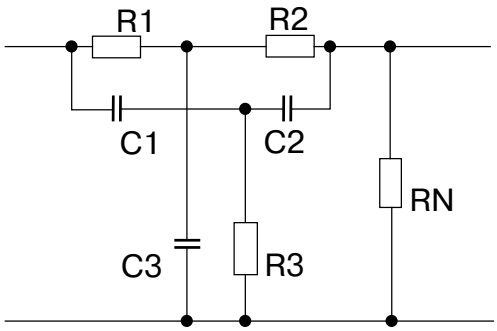


Рис. 2

Вот программа для расчета резонансной частоты фильтра по рис. 1 (автостарт программы со строки 9000, где загружается русский символьный набор):

```
1 BORDER 0: INK 7: PAPER 0: CLS
2 FOR F=0 TO 170 STEP 16: PLOT 0,F: DRAW 255,0: NEXT F
3 FOR F=0 TO 21 STEP 2: PRINT AT F+1,0; (20-F)*8: NEXT F
4 FOR F=54 TO 220 STEP 32: PLOT F,0: DRAW 0,170: NEXT F
5 PRINT AT 0,0;"         40  160 640 2560 10240"
10 INPUT "C1 (ФАРАД) : ";C1: INPUT "C2 (ФАРАД): ";C2: INPUT "R1(ОМ): ";R1: INPUT "R2 (ОМ):
     ";R2
11 LET F=1: LET X=16
12 LET F=F*1.0442738: LET X=X+1
20 LET C1R=1/(F*C1*2*PI)
30 LET C2R=1/(F*C2*2*PI)
40 LET R1C1=R1+C1R
50 LET R2C2=C2R*R2/(C2R+R2)
60 LET Y=R2C2*(170/(R1C1+R2C2))
70 IF X>255 THEN GO TO 10
80 PLOT X,Y
90 GO TO 12
9000 CLEAR 64599: LOAD "Chr" CODE 64600
9010 POKE 23606,88: POKE 23607,251: RUN
```

После старта программы на экран будет выведена сетка с оцифровкой. По горизонтали - частота, по вертикали - затухание фильтра, Введите по очереди номиналы тех элементов, которые будете использовать в фильтре. После ввода всех номиналов программа выдаст Вам амплитудно-частотную характеристику фильтра. Можете оценить ее и попробовать изменить вводимые номиналы для получения требуемой резонансной частоты, добротности и затухания фильтра. При любых изменениях номиналов результаты расчета тут же будут выведены на экран в виде изменившейся характеристики фильтра. Вы имеете возможность сравнивать характеристики между собой, так как все они на одном экране перед Вами. Благодаря этой программе подбор элементов для фильтра с требуемой амплитудно-частотной характеристикой можно выполнить легко и быстро. Результат работы программы представлен на рис. 3.

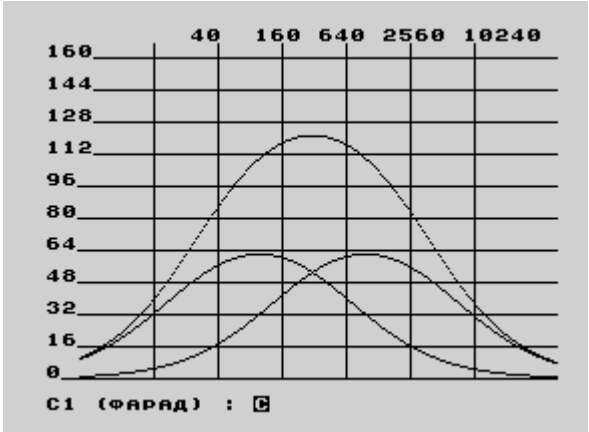


Рис. 3

Для фильтра, представленного на рисунке 2, надо изменить строку 10 программы для ввода большего числа входных параметров:

```
10 INPUT "C1 (ФАРАД): ";C1: INPUT "C2 (ФАРАД): ";C2: INPUT "C3 (ФАРАД): ";C3: INPUT "R1
     (ОМ): ";R1: INPUT "R2 (ОМ): ";R2: INPUT "R3 (ОМ): ";R3: INPUT "R НАГРУЗКИ (ОМ): ";RN
```

Кроме того строки, начиная с 20 должны быть:

```
20 LET A=R2*RN
30 LET B=1/(F*2*PI*C3)
40 LET C=B*A/(B+A)
50 LET V1=C*(170/(R1+C))
60 LET V2=RN*(V1/(R2*RN))
70 LET D=1/(F*2*PI*C2)
80 LET E=D+RN
90 LET G=R3*E/(R3+E)
100 LET H=1/(F*2*PI*C1)
110 LET V3=G*(170/(H+G))
120 LET V4=RN*(V3/(D+RN))
130 IF V2>V4 THEN LET Y=V2: GO TO 150
140 LET Y=V4
150 IF X>255 THEN GO TO 10
160 PLOT X,Y
170 GO TO 12
```

Результат работы программы представлен на рис. 4.

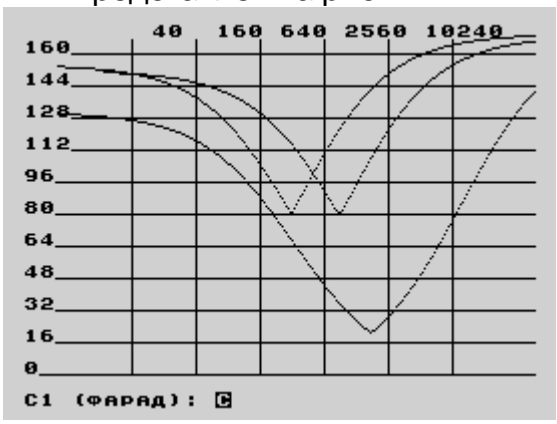


Рис. 4

Если Вас заинтересовали эти программы, то при практической реализации их, мы со своей стороны можем порекомендовать маленькое усовершенствование, которое, наверняка, сможет сделать каждый. Поскольку номиналы конденсаторов удобнее задавать в микрофарадах, а номиналы резисторов - в килоомах, то разумно будет этот пересчет поручить компьютеру. Организуйте для этого дополнительные строки. Кроме того, попробуйте объединить два варианта расчета в одну программу. Для этого можно посоветовать при загрузке готовой программы выводить экран-заставку, на котором будут изображены рисунки 1 и 2, а далее должен следовать запрос, для какой схемы будет производиться расчет.

Нестандартная загрузка.

Комиссаров Павел из пос. Видово Мурманской обл. пишет о том, что он занимается взломом программ, в которых используется нестандартная загрузка или есть что-то необычное. У него уже набралась небольшая коллекция таких загрузчиков. Павел прислал распечатку одного из них. Он загружает блоки кодов с изменением цвета бордюрных полос. Павел пишет, что он взял этот загрузчик из программы "BARBARIAN-3" (BY BILL GILBERT). Он начинается с адреса 65000 и используется точно так же, как и процедура "LOAD BYTES" из ПЗУ, расположенная по адресу 1366 (0556H). Вот этот загрузчик:

```
FDE8   14          INC    D
FDE9   08          EX     AF,AF'
FDEA   15          DEC    D
FDEB   F3          DI
FDEC   3E00        LD     A,#00
FDEE   D3FE        OUT    (#FE),A
FDF0   213F05      LD     HL,#053F
FDF3   E5          PUSH   HL
FDF4   DBFE        IN     A,(#FE)
```

```
FDF6  1F          RRA
FDF7  E620        AND   #20
FDF9  CD6F3C      CALL  #3C6F
FDFC  BF          CP    A
FDFD  00          NOP
FDFE  CD79FE      CALL  #FE79
FE01  30FA        JR    NC,#FDFD
FE03  216900      LD    HL,#0069
FE06  10FE        DJNZ  #FE06
FE08  2B          DEC   HL
FE09  7C          LD    A,H
FE0A  B5          OR    L
FE0B  20F9        JR    NZ,#FE06
FE0D  CD75FE      CALL  #FE75
FE10  30EB        JR    NC,#FDFD
FE12  069C        LD    A,#9C
FE14  CD75FE      CALL  #FE75
FE1T  30E4        JR    NC,#FDFD
FE19  3EC6        LD    A,#C6
FE1B  B0          CP    B
FE1C  30E0        JR    NC,#FDFE
FEIE  24          INC   H
FE1F  20F1        JR    NZ,#FE12
FE21  06C9        LD    B,#C9
FEE3  CD79FE      CALL  #FE79
FE26  30D5        JR    NC,#FDFD
FE28  78          LD    A,B
FE29  FED4        CP    #D4
FE2B  30F4        JR    NC,#FE21
FE2D  CD79FE      CALL  #FE79
FE30  D0          RET   NC
FE31  79          LD    A,C
FE32  EE03        XOR   #03
FE34  4F          LD    C,A
FE35  2600        LD    H,#00
FE37  06B0        LD    B,#B0
FE39  181F        JR    #FE5A
FE3B  08          EX    AF,AF'
FE3C  2007        JR    NZ,#FE45
FE3E  300F        JR    NC,#FE4F
FE40  DD7500      LD    (IX+0),L
FE43  160F        JR    #FE54
FE45  CB11        RL    C
FE47  AD          XOR   L
FE48  C0          RET   NZ
FE49  79          LD    A,C
FE4A  1F          RRA
FE4B  4F          LD    C,A
FE4C  13          INC   DE
FE4D  1807        JR    #FE56
FE4F  DD7E00      LD    A,(IX+0)
FE52  AD          XOR   L
FE53  C0          RET   NZ
FE54  DD23        INC   IX
FE56  1B          DEC   DE
FE57  08          EX    AF,AF'
FE58  06B2        LD    B,#B2
FE5A  2E01        LD    L,#01
FE5C  CB75FE      CALL  #FE75
FE5F  D0          RET   NC
FE60  3ECB        LD    A,#CB
FE62  B8          CP    B
FE53  CB15        RL    L
FE65  0680        LD    B,#B0
FE67  D25CFE      JP    NC,#FE5C
```

```
FE6A  7C        LD    A,H
FE6B  AD        XOR   L
FE6C  67        LD    H,A
FE6D  7A        LD    A,D
FE6E  B3        OR    E
FE6F  20CA      JR    NZ,#FE3B
FE71  7C        LD    A,H
FE72  FE01      CP    #01
FE74  C9        RET
FE75  CD79FE    CALL  #FE79
FE76  D0        RET   NC
FE79  3E16      LD    A,#16
FE7B  3D        DEC   A
FE7C  20FD      JR    NZ,#FE7B
FE7E  A7        AND   A
FE7F  04        INC   B
FE80  C8        RET   Z
FE81  3E7F      LD    A,#7F
FE83  DBFE      IN    A,(#FE)
FE85  1F        RRA
FE86  00        NOP
FE87  A9        XOR   C
FE88  E6E0      AND   #20
FE8A  28F3      JR    Z,#FE7F
FE8C  79        LD    A,C
FE8D  2F        CPL
FE8E  4F        LD    C,A
FE6F  82        ADD   A,D
FE90  A9        XOR   C
FE91  E607      AND   #07
FE93  F608      OR    #08
FE95  D3FE      OUT   (#FE),A
FE97  37        SCF
FE98  C9        RET
```

Перед вызовом загрузчика надо в регистре IX задать адрес загрузки, а в регистре DE - длину загружаемого блока кодов. Процедура вызова загрузчика, например для загрузки экрана-заставки может иметь следующий вид:

```
DD210040    LD    IX,#4000
11001B      LD    DE,#1B00
3EFF        LD    A,#FF
37          SCF
CDE8FD      CALL  #FDE8
C9          RET
```

При этом блок кодов, подлежащий загрузке, может быть записан на ленте обычным способом, надо лишь удалить заголовок, оставив только коды.

Цвет полос бордюра во время загрузки будет периодически меняться, временами полосы вообще будут исчезать, затем появляться вновь с новыми цветами. Клавиша BREAK при загрузке не действует.

По окончании загрузки значение флага переноса содержит информацию об ошибке при загрузке: 1 - не было, 0 - была. Здесь надо сказать, что и как в случае использования процедуры 1366 (0556H) ПЗУ, об ошибке свидетельствует только значение флага переноса, никакие дальнейшие действия, связанные с наличием ошибки не выполняются. Но если Вы хотите, чтобы при ошибке программа останавливалась с выдачей сообщения: "R Tape loading error", то надо поступить аналогично тому, как это делается в ПЗУ. Там для вызова процедуры 1366 используется процедура 2050. Применяя аналогичный прием, вызов приведенного загрузчика может выглядеть следующим образом:

```
...... подготовка пара-
...... метров (см. выше)
CDE8FD      CALL  #FDE8
D8          RET   C
CF          RST   8
1A          DEFB  #1A
```

На этом можно было бы и закончить, но вот что случилось спустя некоторое время, после того, как мы успешно оттестировали этот загрузчик. Неожиданно выяснилось, что загрузчик упорно не хочет работать на другом компьютере. Мы решили выяснить, почему это происходит. К счастью, программа "BARBARIAN-3" оказалась в нашей "фонотеке". Мы опробовали ее и выяснили, что на обоих компьютерах программа нормально загружается. В чем же дело? Стали исследовать коды загрузчика, который приводит Павел. Как и следовало ожидать, загрузчик похож на процедуру "LOAD BYTES", расположенную в ПЗУ по адресу 1366. Но стоп! Что это такое? По адресу FDF9H видим команду: CALL 3C6FH. Это вызов подпрограммы из ПЗУ. Но ведь мы знаем, что в ПЗУ на этом месте ничего нет. Или есть? Так вот, оказывается, где кроется разгадка. Дело в том, что на том компьютере, где этот загрузчик не работал, стоит стандартное ПЗУ, а на том, где работал нормально, стоит ПЗУ "ТУРБО-90", а котором на свободном месте, перед символьным набором расположены программы, расширяющие функциональные возможности компьютера, в частности, загрузка с удвоенной скоростью. Для этого процедура 0556H (1366) этого ПЗУ содержит в точности то, что видим в листинге загрузчика, который приводит Павел, по адресу FDF9H: выполнение подпрограммы 3C6FH. Это связано с загрузкой на удвоенной скорости. В обычном ПЗУ подпрограмма 5C6FH отсутствует, поэтому этот загрузчик и не может работать с другим ПЗУ, кроме "ТУРБО-90". Кстати, у Комиссарова Павла, наверняка, тоже ПЗУ "ТУРБО-90" (необходимый инструмент любого "взломщика").

Вроде бы разобрались, но опять непонятно, почему же программа "BARBARIAN-3" работает с любым ПЗУ: и с обычным, и с "ТУРБО-90". Полезли в программу. А выяснилась очень интересная вещь. В программе "BARBARIAN-3" отсутствует загрузчик кодов в том виде, в каком его приводит Павел. Загрузчик в кодах формируется в программе из подпрограммы 1366 ПЗУ. Мы немного поискали и нашли ту процедуру, которая выполняет эту работу. Она в процессе загрузки "BARBARIAN-3" располагается с адреса C3D5H (50133). Вот что там находится (фрагмент кодов приводим с некоторыми изменениями, в виде завершенной подпрограммы, готовой к употреблению):

```
C3D5  215605      LD    HL,#0556
C3D8  11E8FD      LD    DE,#FDE8
C3DB  01AF00      LD    BC,#00AF
C3DE  EDB0        LDIR
C3E0  2119C4      LD    HL,#C419
C3E3  118FFE      LD    DE,#FE8F
C3E6  010A00      LD    BC,#000A
C3E9  EDB0        LDIR
C3EB  2175FE      LD    HL,#FE75
C3EE  220EFE      LD    (#FE0E),HL
C3F1  22I5FE      LD    (#FE15),HL
C3F4  225DFE      LD    (#FE5D),HL
C3F7  21T9FE      LD    HL,#FE79
C3FA  22FFFD      LD    (#FDFF),HL
C3FD  2224FE      LD    (#FE24),HL
C400  222EFE      LD    (#FE2E),HL
C403  2276FE      LD    (#FE76),HL
C406  215CFE      LD    HL,#FE5C
C409  2268FE      LD    (#FE66),HL
C40C  AF          XOR   A
C40D  32EDFD      LD    (#FDED),A
C410  32FDFD      LD    (#FDFD),A
C413  3286FE      LD    (#FE86),A
C416  C9          RET
C417  00          NOP
C418  00          NOP
C519  82          ADD   A,D
C41A  A9          XOR   C
C41B  E607        AND   #07
C41D  F608        OR    #08
C41F  D3FE        OUT   (#FE),A
C421  37          SCF
C422  C9          RET
```

Эта подпрограмма формирует загрузчик кодов, расположенный с адреса 65000, а далее Вы можете использовать его так, как рассказывалось выше.

Если сформировать загрузчик кодов, пользуясь приведенной подпрограммой C3D5H, при использовании ПЗУ "ТУРБО-90", получим тот загрузчик, который приводит Комиссаров Павел. А если сформировать загрузчик при использовании стандартного ПЗУ, то получим следующие отличия по-сравнению с вариантом для "ТУРБО-90":

```
FDF9   F602        OR     #02
FDFB   4F          LD     C,A
FE03   211504      LD     HL,#0415
```

Кстати, если внести эти изменения в вариант Павла, то загрузчик будет работать с обоими ПЗУ. Почему же в "BARBARIAN-3" используется казалось бы более головоломный прием, когда загрузчик формируется при помощи другой подпрограммы (C3D5H) из процедуры 0556H ПЗУ? Ответ прост. Если у Вас стандартное ПЗУ, то Вы работаете как обычно. А если у Вас "ТУРБО-90", то сохранится возможность загружать всю игру в режиме с удвоенной скоростью (как это предусмотрено, введя "-" перед командой LOAD ""). Несмотря на то, что загрузчик кодов в общем-то нестандартный, в нем поддерживается турбо-режим. Это происходит как раз из-за вызова подпрограммы 3C6FH, который мы видим по адресу FDF9H. Те читатели, у которых есть "ТУРБО 90", могут убедиться в том, что загрузчик, приведенный Павлом, работает и на удвоенной скорости (правда при этом пропадает эффект переключения цвета бордюрных полос). Кроме того, процедура, которая создает загрузчик, гораздо короче самого загрузчика, поэтому такой путь еще и экономит память и время загрузки исходной программы.

Эта история показывает, насколько осторожным и предусмотрительным надо быть, проектируя автономные загрузчики для загрузки кодов. Иначе за Вашей замечательной программой, в которой Вы примените такой загрузчик, может закрепиться репутация "капризной" или "неработоспособной". Надеемся все же, что приведенный пример не отпугнет программистов, а вооружит сведениями, позволяющими выполнить эту работу более квалифицированно.

ПРОФЕССИОНАЛЬНЫЙ ПОДХОД

Отладчик машинного кода "TRACER"

Создание резидентных программ - один из самых интересных моментов в работе программиста. Резидентные программы работают как бы "поверх" основных и позволяют не выходя из главной программы выполнять еще какую-либо ценную задачу. Так, например, в разделе "ФОРУМ" Вам была представлена резидентная программа-русификатор, которая 50 раз в секунду проверяет нажатие комбинации клавиш SYM.SH. + ENTER и, если они не нажаты, то резидент себя не проявляет и остается "прозрачным" для пользователя.

В игровых программах резидентные процедуры как правило занимаются тем, что воспроизводят музыку одновременно с работой других процедур, а иногда им поручают обслуживание клавиатуры в поисках нажатия нужной клавиши. Нередко их используют и при организации мультипликации.

Создание резидентных программ для "Спектрума" основывается на использовании режима прерываний 2-го рода (IM2) и сегодня мы представляем Вашему вниманию еще одну полезную резидентную программу, которая поможет тем, кто программирует в машинных кодах, особенно там, где иные средства откажутся работать.

Кто не испытывал трудностей при отладке программ в машинных кодах, когда казалось бы вполне "хорошая" программа почему-то вдруг ни с того ни с сего "зависает" и никак не удается определить, почему же это происходит. Проблему отчасти решает пошаговая отладка программы при помощи мощных мониторов, таких, как, например "MONS-3", "MONS-4". Но только отчасти, так как для того, чтобы вручную "прощелкать" достаточно большую программу, требуется уйма времени, ведь программы содержат массу различных циклов и придется повторять выполнение программы в одних и тех же адресах многократно. В общем задача эта - не из простых. Пошаговая отладка - это конечно, хорошо, но неужели только этим ограничиваются возможности "влезть внутрь" программы и посмотреть, что же там происходит. Сегодня мы предлагаем читателям еще один инструмент, позволяющий заглянуть внутрь программы во время ее работы, причем в режиме реального времени. Конечно, использование приведенного метода требует определенной квалификации, но если Вы хорошо разберетесь с ним, то получите в свой арсенал инструмент, значение которого трудно переоценить.

Итак, вниманию читателей предлагается отладчик машинного кода, использующий режим прерываний второго рода.

Программа-отладчик машинного кода называется "TRACER". Она позволяет выводить в правой части экрана заданную информацию в шестнадцатеричном коде. Что это может быть за информация? Это зависит от Ваших требований. Что Вы зададите. Например, Вы хотите просмотреть число, записанное в какой-нибудь ячейке памяти, как оно изменяется. Или как меняется содержимое какого-то регистра. Или что записано на вершине машинного стека. Принцип работы отладчика состоит в том, что 50 раз в секунду будет происходить прерывание Вашей исследуемой программы и 50 раз в секунду интересующие Вас сведения будут выведены на экран в столбец в правой части экрана. Процедура, обслуживающая прерывание, должна формировать требуемый параметр и выдавать его на печать. От того, что конкретно Вы хотите просмотреть, зависит начало этой процедуры. В качестве примера, рассмотрим вариант кодов, который позволяет просмотреть число, записанное на вершине машинного стека. Блок кодов, который это делает, приведен в распечатке (Листинг 1), а возможный пример работы отладчика - на рис. 1.

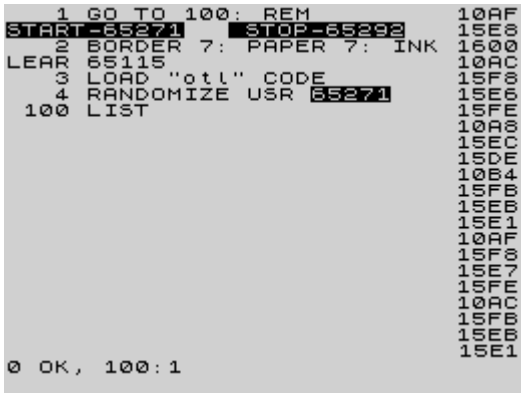


Рис. 1

Включение отладчика в работу происходит при выполнении инициализирующей подпрограммы START, расположенной с адреса FEF7H (RANDOMIZE USR 65271). При этом происходит следующее. Вначале задается адрес в дисплейном файле, куда будет производиться вывод результатов: в регистр HL записывается адрес верхней пиксельной линии того знакоместа экрана, куда будет производиться печать. Это значение заносится в системную ячейку FF13H. Затем обнуляется другая системная ячейка 5C81H (это неиспользуемый адрес в таблице системных переменных). Здесь будет организован счетчик строк для выводимой информации. После этого происходит включение режима прерываний второго рода. В регистр I процессора записывается число FEH. Поэтому при прерывании произойдет переход на программу, адрес которой записан в ячейке памяти FEFFH. Там записан адрес FE5CH. Это начало обрабатывающей процедуры. Затем происходит переключение на режим прерываний второго рода и возврат.

Теперь 50 раз в секунду будет выполняться обслуживающая процедура. По условию, которое мы задали, она должна выдавать значение двухбайтного числа, находящегося на вершине машинного стека. Считывание этого значения происходит следующим образом. Текущее значение регистра HL сохраняется в неиспользуемой ячейке 6CB0H в таблице системных переменных. Далее интересующее нас число с вершины стека заносится в регистр HL. При этом необходимо восстановить стек в том виде, в каком он был, что выполняется при помощи обратной записи числа из HL на стек. Далее на стеке сохраняются значения всех регистров процессора, которые задействованы и будут изменены в процедуре обработки прерывания.

Теперь надо интересующее нас число (в регистре HL) вывести на печать. Это выполняется следующим образом. Проверяется значение счетчика строк 5C81H. Если не достигнут нижний край экрана, то происходит переход на процедуру CONT, а если достигнут, то для последующего вывода задается опять верхняя строка экрана путем занесения в системный счетчик FF13H исходного адреса в дисплейном файле. Счетчик строк при этом обнуляется.

Процедура CONT начинается с того, что в регистр DE наносится текущее значение адреса для вывода в дисплейном файле. Затем старший байт интересующего числа записывается в регистр A и при помощи ротации вправо старший полубайт становится на место младшего. Выполняется его печать при помощи подпрограммы PRINT, которая выполняет печать четырех младших битов, содержащихся в регистре A (в шестнадцатеричном представлении четырем битам соответствует один символ). После печати значение DE увеличивается на единицу, что соответствует переходу к следующему знакоместу. Затем в регистр A опять записывается значение из регистра H и выполняется печать младшего полубайта. Значение DE опять увеличивается на 1. Далее то же повторяется и со значением регистра L, точно так же за два приема выполняется печать двух полубайтов.

Далее, путем выполнения ротации и суммирования, в счетчике FF13H получается следующий адрес в дисплейном файле, соответствующий выводу на следующей строке. Такой способ вычисления этого адреса связан с тем, что экранная память состоит из трех сегментов и надо получить правильный результат при переходе от одного сегмента к

другому. В нашей книге "Элементарная графика" мы подробно разбирали все вопросы, связанные с выводом на экран. Поэтому, для более ясного представления о том, как все это происходит, заказывайте и читайте нашу книгу.

Листинг 1

```
FE5C  E2B05C  INT S/R  LD    (#5CB0),HL      ;Сохранение HL.
FE5F  E1               POP   HL              ;Получение требуемого
                                             ;параметра в HL.
FE60  E5               PUSH  HL              ;Восстановление стека.
FE61  F5               PUSH  AF              ;Сохранение
FE62  C5               PUSH  BC              ;значений
FE63  D5               PUSH  DE              ;регистров.
FE64  3A815C           LD    A,(#5C81)       ;Переход к
FE67  3C               INC   A               ;следующей
FE68  32815C           LD    (#5C81),A       ;строке.
FE6B  FE16             CP    #16             ;Проверка на достижение
FE6D  200B             JR    NZ,#FE7A        ;конца экрана.
FE6F  111C40           LD    DE,#401C        ;Переход на
FE72  ED5313FF         LD    (#FF13),DE      ;начало экрана.
FE76  AF               XOR   A               ;Обнуление
FE77  32815C           LD    (#5C81),A       ;счетчика строк

FE7A  ED5B13FF   CONT  LD    DE,(#FF13)      ;Текущий адрес
                                             ;в дисплейном файле.
FE7F  7C               LD    A,H             ;Выделение
FE7F  1F               RRA                   ;старшего
FE80  1F               RRA                   ;полубайта.
FE81  1F               RRA                   ;
FE82  1F               RRA                   ;
FE83  CDB9FE           CALL  #FEB9           ;Печать старшего
FE86  13               INC   DE              ;полубайта.
FE87  7C               LD    A,H             ;
FE88  CDB9FE           CALL  #FEB9           ;Печать младшего
FE8B  13               INC   DE              ;полубайта.
FE8C  7D               LD    A,L             ;Выделение
FE8D  1F               RRA                   ;старшего
FE8E  1F               RRA                   ;полубайта.
FE8F  1F               RRA                   ;
FE90  1F               RRA                   ;
FE91  CDB9FE           CALL  #FEB9           ;Печать старшего
FE94  13               INC   DE              ;полубайта.
FE95  7D               LD    A,L             ;
FE96  CDB9FE           CALL  #FEB9           ;Печать мл. полубайта.
FE99  2A13FF           LD    HL,(#FF13)      ;Расчет
FE9C  CB1C             RR    H               ;адреса
FE9E  CB1C             RR    H               ;в дисплейном
FEA0  CB1C             RR    H               ;файле
FEA2  012000           LD    BC,#0020        ;с учетом
FEA5  ED4A             ADC   HL,BC           ;сегмента
FEA7  CB14             RL    H               ;экрана.
FEA9  CB14             RL    H               ;
FEAB  CB14             RL    H               ;
FEAD  2213FF           LD    (#FF13),HL      ;
FEB0  D1               POP   DL              ;Финишные операции
FEB1  C1               POP   BC              ;по восстановлению
FEB3  F1               POP   AF              ;значений
FEB3  2AB05C           LD    HL,(#5CB0)      ;регистров.
FEB6  C33800           JP    #0038           ;Переход на RST #38

FEB9  E60F       PRINT AND   #0F               ;Преобразование
FEBB  87               ADD   A,A              ;кода в символ
FEBC  E5               PUSH  ML               ;согласно
FEBD  21D7FE           LD    HL,#FED7         ;таблице
```

```
FEC0  0600            LD    B,#00          ;с учетом
FEC2  4F              LD    C,A            ;шестнадцати-
FEC3  09              ADD   HL,BC          ;ричного
FEC4  46              LD    B,(HL)         ;представления.
FEC5  23              INC   HL             ;
FEC6  4E              LD    C,(HL)         ;
FEC7  C5              PUSH  BC             ;
FEC8  E1              POP   HL             ;
FEC9  0608            LD    B,#08          ;
FECB  7E        PRT   LD    A,(HL)         ;Вывод
FECC  12              LD    (DE),A         ;символа
FECD  23              INC   HL             ;на
FECE  14              INC   D              ;экран.
FECF  10FA            DJNZ  #FECB          ;
FED1  7A              LD    A,D            ;
FED2  D608            SUB   #08            ;
FED4  57              LD    D,A            ;
FED5  E1              POP   HL             ;
FEDS  C9              RET                  ;

FED7  3D803D    TABLE DEFB                 ;Таблица
FEDA  883D90          DEFB                 ;для
FEDD  3D983D          DEFB                 ;преобразования
FEE0  A03DA8          DEFB                 ;кода
FEE3  3DB03D          DEFB                 ;в символ
FEE6  B83DC0          DEFB                 ;с учетом
FBE9  3DC83E          DEFB                 ;шестнадцати-
FEEC  083E10          DEFB                 ;ричного
FEEF  3E183E          DEFB                 ;представления.
FEF2  203E28          DEFB                 ;
FEF5  3E30            DEFB                 ;

FEF7  211C40    START LD    HL,#401C       ;Инициализирующая
FEFA  2213FF          LD    (#FF13),HL     ;процедура
FEFD  1802            JR    #FF01

FEFF  5CFE      ADR   DEFB                 ;Адрес перехода по IM2.

FF01  AF        PASS  XOR   A              ;Продолжение
FF02  32815C          LD    (#5C81),A      ;инициализирующей
FF05  3EFE            LD    A,#FE          ;процедуры.
FF07  ED47            LD    I,A            ;
FF09  ED5E            IM    2              ;
FF0B  C9              RET                  ;

FF0C  ED56      STOP  IM    1              ;Восстанавливающая
FF0E  3E3F            LD    A,#3F          ;процедура.
FF10  ED47            LD    I,A            ;
FF12  C9              RET                  ;

FF13  00        SCRP  NOP                  ;Текущее значение адреса
FF14  00              NOP                  ;в дисплейном файле.
```

После выполнения печати искомой величины, происходят финишные операции. Это восстановление со стека значений регистров процессора. При этом значение регистра HL, как мы помним, находится в ячейке 5CD0H, оттуда его и считываем. Далее - переход на адрес 0038H, как и в случае прерывания первого рода.

Непосредственно печать выполняется при помощи подпрограмм печати PRINT и PRT. Вначале зануляются старшие байты регистра A (выполняется печать только одного полубайта). Для того, чтобы обеспечить печать в шестнадцатеричном виде, используется перевод кода в символ при помощи таблицы, расположенной с адреса FED7H. В зависимости от значения кода происходит установка в регистре HL адреса в символьном наборе, с которого находится изображение этого символа. Затем процедура PRT выполняет

переписывание восьми байт из символьного набора в дисплейный файл.

В результате работы программы-отладчика в правой части экрана Вы будете видеть столбец непрерывно меняющихся цифр. Это и есть результат работы программы - числа, которые находятся на вершине стека в процессе работы программы. Они меняются достаточно быстро, но ведь все происходит в режиме реального времени. Да и к тому же это просто демонстрационный пример, показывающий возможности программы.

На практике, для того, чтобы "вытаскивать" из исследуемой программы требуемые параметры, Вам придется несколько видоизменить начальную часть процедуры, обслуживающей прерывание. Хорошенько разобравшись с принципом действия этой программы, Вы сможете легко применять ее для своих вполне конкретных целей.

Для того, чтобы отключить отладчик, надо выполнить останавливающую подпрограмму, расположенную с адреса FF0CH (RANDOMIZE USR 65292). При этом происходит переключение на режим прерываний 1 рода - восстановление обычного режима работы "Спектрума".

И в заключении об одном ограничении на работу отладчика. Вывод на экран будет возможен только в том случае, если прерывания разрешены (EI). Если в исследуемой программе они запрещаются командой DI, то прерываний не будет и, следовательно, не будет вывода на экран.

МАЛЕНЬКИЕ ХИТРОСТИ

ПРОГРАММА "ТЕНЕВОЙ КОНТУР"

На страницах "ZX-РЕВЮ" уже не раз приводились примеры того, как можно внести некоторые усовершенствования в программы на Бейсике, применяя для этого фрагменты в машинных кодах. Сегодня, в продолжение этой темы, мы приводим блок кодов, который позволит вносить определенные изменения в изображение, имеющееся на экране. Посмотрите на рисунок. Изображение, которое Вы видите, как бы объемно. Создается впечатление, что оно приподнято над плоскостью экрана. Вам, наверное, приходилось видеть такой прием в программах. Мы предлагаем Вашему вниманию оригинальный способ получения этого эффекта.

Суть метода заключается в том, что теневое оконтуривание выполняется автоматически программой в машинных кодах. Сначала "плоское" изображение выводится на экран обычными способом, будь то Бейсик или процедуры в машинных кодах.

Можно даже заготовить "плоскую" картинку при помощи графического редактора. В общем, представим, что имеется изображение, полученное любым способом. Затем, вызывая приведенную ниже программу в кодах, выполняется теневое оконтуривание этого изображения, причем весь экран может быть обработан за один прием, а можно вызывать кодовую программу несколько раз, выполняя оконтуривание в несколько приемов.

Критерием, определяющим то, что надо оконтуривать, а что не надо, является повышенная яркость (BRIGHT 1). Все, что на экране нарисовано с повышенной яркостью, будет оконтурено при вызове блока в машинных кодах. Как это эффективнее всего использовать в программах придумайте сами. Дело за Вашей фантазией. Мы же предлагаем программу в машинных кодах под названием "Теневой контур". Ее БЕЙСИК-загрузчик приведен в Листинге_1, а машинный код с комментариями - в Листинге_2.

Листинг_1

```
10 FOR i = 1 TO 211
20 READ a
30 POKE 63999+i, a
40 NEXT i
50 DATA   243,   253,   229,   253,   33
60 DATA   255,   90,    6,     24,    197
70 DATA   6,     32,    197,   253,   126
80 DATA   0,     254,   64,    56,    15
90 DATA   253,   126,   1,     254,   64
100 DATA  220,   82,    250,   253,   126
110 DATA  32,    254,   64,    56,    93
120 DATA  253,   43,    193,   16,    228
130 DATA  193,   16,    222,   253
140 DATA  225,   251,   201,   122,   230
150 DATA  3,     7,     7,     7,     246
160 DATA  64,    103,   107,   201,   36
170 DATA  124,   230,   7,     192,   124
```

```
180 DATA  214,   8,     103,   125,   198
190 DATA  32,    111,   208,   124,   198
200 DATA  8,     103,   238,   88,    192
210 DATA  38,    0,     201,   120,   254
220 DATA  32,    200,   253,   229,   209
230 DATA  19,    205,   47,    250,   221
240 DATA  33,    191,   250,   6,     8
250 DATA  62,    64,    253,   190,   33
260 DATA  56,    9,     6,     10,    253
270 DATA  190,   32,    48,    2,     6
280 DATA  12,    221,   126,   0,     182
290 DATA  119,   221,   35,    205,   58
300 DATA  250,   16,    244,   201,   225
310 DATA  241,   245,   229,   254,   24
320 DATA  210,   35,    250,   253,   229
330 DATA  209,   235,   1,     32,    0
340 DATA  9,     235,   14,    2,     221
350 DATA  33,    203,   250,   205,   47
360 DATA  250,   6,     4,     221,   126
370 DATA  0,     182,   119,   221,   35
380 DATA  205,   58,    250,   16,    244
390 DATA  253,   126,   33,    254,   64
400 DATA  210,   35,    250,   241,   245
410 DATA  254,   32,    210,   35,    250
420 DATA  19,    13,    32,    220,   195
430 DATA  35,    250
500 DATA  0,     0,     128,   64,    160
510 DATA  80,    160,   80,    160,   80
520 DATA  160,   80
600 DATA  42,    21,    10,    5,     128
610 DATA  64,    160,   80
```

Для демонстрации действия блока машинных кодов надо набрать Бейсик-программу, приведенную ниже. Программа имеет автостарт со строки 2, где происходит загрузка необходимых кодов. Сам блок "теневой контур" надо оформить в виде файла "ten" CODE 64000,211. Кроме того, программа русифицирована по методике, приведенной в ZX-РЕВЮ-92 N1,2. стр.31. Русско-латинский символьный набор загружается в виде файла "chr" CODE 64600,768.

ЛИСТИНГ_2

```
FA00  F3                  DI                ;
FA01  FDE5                PUSH  IY          ;
FA03  FD21FF5             LD    IY,#5AFF    ;Последний байт
                                            ;файла атрибутов.
FA07  0616                LD    B,#16       ;Организация цикла
FA09  C5          LOOP    PUSH  BC          ;для 22 строк.
FA0A  C620                LD    B,#20       ;Организация цикла
FA0C  C5          LOOP_1  PUSH  BC          ;для 32 колонок.
FA0D  FD7E00              LD    A,(IY+0)    ;Наличие BRIGHT 1
FA10  FE40                CP    #40         ;в текущем знакоместе.
FA12  380F                JR    C,#FA23     ;Если нет, то переход
                                            ;на PASS.
FA14  FD7E01              LD    A,(IY+1)    ;Крайнее ли это знако-
FA17  FE40                CP    #40         ;место с BRIGHT 1.
FA19  DC52FA              CALL  C,#FA52     ;Если да, то выполнение
                                            ;вертикального
                                            ;оконтуривания - VERT.
FA1C  FD7E20              LD    A,(IY+32)   ;Если под текущим знако-
FA1F  FE40                CP    #40         ;местом не BRIGHT 1, то
FA21  385D                JR    C,#FA80     ;выполнение горизонталь-
                                            ;ного оконтуривания HORIZ
FA23  FD2B        PASS    DEC   IY          ;Финишная процедура, если
```

```
FA25  C1                    POP   BC              ;знакоместо не с BRIGHT 1
FA26  10E4                  DJNZ  #FA0C           ;
FA28  C1                    POP   BC              ;
FA29  10DE                  DJNZ  #FA09           ;
FA2B  FDE1                  POP   IY              ;
FA2D  FB                    EI                    ;
FA2E  C9                    RET                   ;

FA2F  7A          ADRES     LD    A,D             ;Эта подпрограмма выполня-
FA30  E603                  AND   #03             ;ет расчет адреса в дисп-
FA32  07                    RLCA                  ;лейном файле по известно-
FA33  07                    RLCA                  ;му адресу в файле атрибу-
FA34  07                    RLCA                  ;тов.
FA35  F640                  OR    #40             ;
FA37  67                    LD    H,A             ;
FA38  68                    LD    L,E             ;
FA39  C9                    RET                   ;

FA3A  24          OVER      INC   H               ;Эта подпрограмма выполня-
FA3B  7C                    LD    A,H             ;ет операции, связанные с
FA3C  E607                  AND   #07             ;наложением оконтуривающе-
FA3E  C0                    RET   NZ              ;го изображения на тот ри-
FA3F  7C                    LD    A,H             ;сунок, который имеется на
FA40  D608                  SUB   #08             ;экране.
FA42  67                    LD    H,A             ;
FA43  7D                    LD    A,L             ;
FA44  C620                  ADD   A,#20           ;
FA46  6F                    LD    L,A             ;
FA47  D0                    RET   NC              ;
FA48  7C                    LD    A,H             ;
FA49  0606                  ADD   A,#08           ;
FA4B  67                    LD    H,A             ;
FA4C  EE58                  XOR   #58             ;
FA4E  C0                    RET   NZ              ;
FA4F  2600                  LD    H,#00           ;
FA51  C9                    RET                   ;

FA52  78          VERT      LD    A,B             ;Не последняя ли это коло-
FA53  FE20                  CP    #20             ;нка, если да, то возврат,
FA55  C8                    RET   Z               ;т.к. негде оконтуривать.
FA56  FBE5                  PUSH  IY              ;
FA58  D1                    POP   DE              ;
FA59  13                    INC   DE              ;
FA5A  CD2FFA                CALL  #FA2F           ;
FA5D  DD21BFFA              LD    IX,#FABF        ;Базовый адрес таблицы
                                                  ;для построения оконтури-
                                                  ;вающего рисунка.
FA61  0608                  LD    B,#08           ;
FA63  3E40                  LD    A,#40           ;
FA65  FDBE21                CP    (IY+33)         ;Если знакоместо не угло-
FA68  3809                  JR    C,#FA73         ;вое, то переход на LOOP_2
FA6A  060A                  LD    A,#0A           ;если угловое, то оконту-
FA6C  FDBE20                CP    (IY+32)         ;ривание снизу.
FA6F  3002                  JR    NC,#FA73        ;
FA71  060C                  LD    B,#0C           ;
FA73  DD7E00      LOOP_2    LD    A,(IX+0)        ;Проверка необходимости
FA76  B6                    OR    (HL)            ;выполнения наложения
FA77  77                    LD    (HL),A          ;на имеющееся на экране
FA78  DD23                  INC   IX              ;изображение.
FA7A  CD3AFA                CALL  #FA3A           ;Выполнение наложения.
FA7D  10F4                  DJNZ  #FA73           ;
FA7F  C9                    RET                   ;

FA80  E1          HORIZ     POP   HL              ;
FA81  F1                    POP   AF              ;
```

```
FA82  F5                      PUSH  AF                ;
FA83  E5                      PUSH  HL                ;
FA84  FE18                    CP    #18               ;Не последняя ли строка.
FA86  D223FA                  JP    NC,#FA23          ;Если да, то негде окон-
                                                      ;туривать и переход на
                                                      ;PASS.
FA89  FDE5                    PUSH  IY                ;
FA8B  D1                      POP   DE                ;
FA8C  EB                      EX    DE,HL             ;
FA8D  012000                  LD    BC,#0020          ;
FA90  09                      ADD   HL,BC             ;
FA91  EB                      EX    DE,HL             ;
FA92  0E02                    LD    C,#02             ;
FA94  DD21CBFA                LD    IX,#FACB          ;Базовый адрес в таблице,
                                                      ;определяющей оконтурива-
                                                      ;ющий рисунок.
FA98  CD2FFA                  CALL  #FA2F             ;
FA9B  0604                    LD    B,#04             ;
FA9D  DD7E00                  LD    A,(IX+0)          ;Проверка необходимости
FAA0  B6                      OR    (HL)              ;выполнения наложения
FAA1  77                      LD    (HL),A            ;на имеющееся на экране
FAA2  DD23                    INC   IX                ;Изображение.
FAA4  CD3AFA                  CALL  #FA3A             ;Выполнение наложения.
FAA7  10F4                    DJNZ  #FA9D             ;
FAA9  FD7E21                  LD    A,(IY+33)         ;
FAAC  FE40                    CP    #40               ;
FAAE  D223FA                  JP    NC,#FA23          ;
FAB1  F1                      POP   AF                ;
FAB2  F5                      PUSH  AF                ;
FAB3  FE20                    CP    #20               ;
FAB5  D223FA                  JP    NC,#FA23          ;
FAB8  13                      INC   DE                ;
FAB9  0D                      DEC   C                 ;
FABA  20DC                    JR    NZ,#FA98          ;
FABC  C323FA                  JP    #FA23             ;
FABF  000080      PICTURE     DEFB  #00,#00,#80       ;Базовая таблица, по ко-
FAC2  40A050                  DEFB  #40,#A0,#50       ;торой строится оконтури-
FAC5  A050A0                  DEFB  #A0,#50,#A0       ;вающий рисунок.
FAC8  50A050                  DEFB  #50,#A0,#50       ;
FACB  2A150A                  DEFB  #2A,#15,#0A       ;
FACE  058040                  DEFB  #05,#80,#40       ;
FAD1  A050                    DEFB  #A0,#50           ;
```

```
 1 GO TO 100
 2 CLEAR 63999: LOAD "ten" CODE 64000
 3 LOAD "chr" CODE 64600
 4 POKE 23606,88: POKE 23607,251: REM RUS
 100 BORDER 1: PAPER 1: INK 7: BRIGHT 0: CLS
 200 BRIGHT 1
 300 LET A$="B"
 1000 FOR Y=8 TO 12: FOR N=0 TO 15: PRINT AT Y, 1+N;A$;: NEXT N: NEXT Y
 1010 FOR Y=3 TO 17: FOR N=0 TO 9: PRINT AT Y, 4+N;A$: NEXT N: NEXT Y
 1020 FOR Y=5 TO 15: PRINT BRIGHT 0;AT Y,7;"    ":NEXT Y
 1030 PRINT BRIGHT 0;AT 10,3;"            "
 2000 FOR N=5 TO 15: PRINT AT N, 18;"              ": NEXT N
 2010 PRINT AT 8, 20; "ПРОГРАММА"
 2020 PRINT AT 9, 20; "ДЕМОНСТРИ-"
 2030 PRINT AT 10, 22; "РУЮЩАЯ"
 2040 PRINT AT 12, 20; """ТЕНЕВОЙ"
 2050 PRINT AT 13, 23; "КОНТУР"""
 2100 PAUSE 50: PLOT 145,48: DRAW 0,86: DRAW 102,0
 3000 PAUSE 50: RANDOMIZE USR 64000
 4000 BRIGHT 0
```

После старта программы со строки 2, загрузки кодовых блоков и включения русского символьного набора, происходит формирование изображения на экране, которое в дальнейшем будет оконтурено. Для этого перед подачей команд PRINT происходит включение режима повышенной яркости BRIGHT 1 (строка 200). Обратите взимание, что отверстие в левой фигуре, состоящей из букв "B", прорисовывается в режиме BRIGHT 0. В правой части экрана получен прямоугольник, подлежащий оконтуриванию. Это выполняется печатью нескольких строк пробелов в режиме BRIGHT 1 (строка 2000), а текст впечатывается уже потом. Для завершенности фигуры этот прямоугольник слева и сверху замкнут прорисовкой прямых линий (строка 2100), так как оконтуривание производится только справа и снизу.

Если оконтуривание должно быть выполнено поверх символов, напечатанных на экране и имеющих BRIGHT 0, то происходит наложение оконтуривающего фона на имеющееся изображение, это можно наблюдать, если добавить в программу строку:

```
150 FOR N=1 TO 32*22: PRINT "+"; : NEXT N
```

Итак, дело за Вашей фантазией, уважаемые читатели!

Компьютерная новелла

Дорогие читатели!

С этого номера мы начинаем новую постоянную рубрику в "ZX-РЕВЮ" для тех, кто любит и ценит игровые программы. Эту рубрику мы назовем "Компьютерная новелла".

Изобретатель этой рубрики, он же ее главный редактор - наш постоянный корреспондент из Москвы Матвеев Юрий Александрович. Он экспериментирует в этом жанре уже не первый год и сегодня на Ваш суд предлагается первая работа. В ближайших выпусках Вас ожидают новые увлекательные приключения героев широкоизвестных компьютерных игр.

Должны сказать, что за прошедшие годы мы не раз пробовали экспериментировать с представлением игровых программ. Ключевой проблемой здесь является, на наш взгляд то, что подробная "раскрутка" игры и доведение ее до уровня простейших подсказок (hints) не только не способствует ее популяризации, но и наоборот, навсегда может оттолкнуть от нее возможного пользователя. Как Вы думаете, что скажут альпинисты, если подходы к основным вершинам мира кто-то услужливо забетонирует ступеньками?

Итак, основная цель хорошо составленного описания компьютерной игры должна быть, на наш взгляд в том, чтобы с одной стороны дать заинтересованному пользователю необходимые "хинты" для прохождения игры, но, с другой стороны, сделать это настолько мягко, чтобы они не бросались в глаза при первом прочтении и не портили бы будущую игру.

Так родилась идея компьютерной новеллы. Идея понравилась нам еще и потому, что такую новеллу могут читать отнюдь не только те, кто имеют данную программу и желают ее пройти. В принципе, она делается как независимое художественное произведение и может читаться даже теми, у кого нет и не было под рукой персонального компьютера. И если у них появится после этого желание приобрести компьютер, найти и запустить эту игру, вот тогда мы и будем считать, что наша главная цель - достигнута.

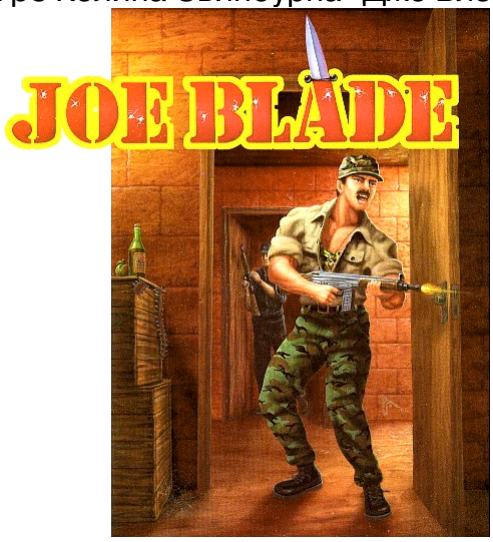
У Вас на глазах рождается новый жанр. Мы внимательно следим за новинками в зарубежных компьютерных журналах, но систематического исследования такого направления пока не встречали, т.е. дело это совершенно неисследованное. Вы можете и сами принять посильное участие в становлении и обкатке этого направления. Мы будем признательны за критические отзывы, советы, рекомендации. Может быть у Вас есть какие-то иные оригинальные идеи развития такого жанра. Мы с радостью примем Ваши пожелания, а наиболее интересные и дискуссионные обсудим на страницах "ZX-РЕВЮ". Письма можете направлять по нашему адресу на имя редактора раздела "К.Н." - Матвеева Ю.А.

* * *

Матвеев Ю. А.

КОНФЕРЕНЦИЯ

(по игре Колина Свинбурна "Джо Блэйд").



I

Джо Блэйд, удобно устроившись на заднем сидении автомобиля, прикурил сигарету и вопросительно посмотрел на мэра города.

- Едем, - сказал тот шоферу и, когда они выехали на шоссе, ведущее к городской тюрьме, наконец повернулся к Джо:

- Ты смелый парень и мне остается надеяться только на тебя. - Он сделал многозначительную паузу. - Ты, конечно, знаешь, кто такой Кракс Бладфингер: влиятельное лицо в городе, президент мощной военно-промышленной компании, почетный член городского магистрата.

Джо усмехнулся. Он никогда не признавал авторитетов, тем более тех, кто стоял у власти. Власть, по мнению Джо Блэйда, всегда ограничивала его свободу, мешала спокойно жить, а свободу он любил больше всего.

Мэр расценил усмешку собеседника как иронию и поспешил согласиться.

- Да, ты прав. Этот человек - оборотень, и не тот, за кого себя выдает. В полиции уже давно под него копают, но... Его организация словно спрут охватила важнейшие точки города. Везде есть его люди. По сути, мэр города - это он. Я - лишь прикрытие и ничего не могу сделать. Все идет так, как хочет он. О чем говорить, если даже городская тюрьма в его подчинении. Есть сведения, что Кракс превратил ее в склад боеприпасов. Также известно, что Кракс Бладфингер нелегально торгует не только оружием, но и наркотиками. У него четко отлаженная сеть сбыта и верная агентура.

Пока мэр рассказывал о Краксе, Джо Блэйд, приоткрыв окно, щелкнул в туман окурком. Мэр не открыл Америки, в народе уже давно ходили слухи о мафии Кракса, однако на официальном уровне это никогда не обсуждалось.

- Я так понимаю, - задумчиво отозвался Джо, - что на сегодняшний день вопрос встал ребром: или вы, или Кракс?

- Не совсем, - ответил мэр. - Все дело в том, что завтра здесь начинается международная конференция по правам человека. Это, можно сказать, крупнейшее событие в истории нашего города. Среди прочих к нам вчера приехали шесть лидеров мирового ранга. В какой-то степени они могут изменить положение. Их влияние в мире, их связи были бы способны очистить наш город от этого паука Кракса и его нечисти. Однако...

- Однако..? - повторил Джо Блэйд.

Вчера ночью Кракс похитил их и теперь шантажирует меня.

- Что? Всех шестерых?

- Да. Впрочем это, наверное, от жадности или от наглости. Для скандала достаточно

было бы и одного.

- Так чего же он хочет?

- Завтра на Конференции я, как мэр города, делаю доклад. Я собирался, когда готовил речь, дать понять о мощной мафиозной структуре, выросшей в нашем городе за последнее время. Я хотел косвенно указать на Кракса, как на главаря этой структуры. И потом…

- Интерпол..? - засмеялся Джо Блэйд.

- Вот именно. Но рукопись моей речи каким-то образом оказалась у Кракса. И он среагировал быстро. Возможно, ему было бы проще устроить мне автомобильную аварию, но я предусмотрительно усилил охрану и стал для него недоступен. Я даже подумать не мог, что он решится на захват заложников. Сегодня Кракс звонил мне и намеком дал понять, что если я делаю доклад в его пользу…

- То есть? - переспросил Джо.

- …Кракс Бладфингер первое лицо в городе, отстаивающее права человека, борющееся с преступностью, безгранично верное Президенту и стране, гений, ну и так далее. Он хочет добиться благосклонности всех мировых лидеров, которые будут присутствовать на конференции. Он ждет аплодисментов, но он их не получит! - Мэр сжал кулаки.

- Конечно, он же показал теперь всем свое истинное лицо, - улыбнулся Джо Блэйд.

- Нет. Просто вчера, ближе к ночи, сразу после приезда делегатов, он предложил им в рамках ознакомления побывать в городской тюрьме, посмотреть условия жизни заключенных, уровень охраны… До сих пор они не вернулись. Кракс придумал предлог, чтобы задержать их там до начала конференции и только после моего доклада в его пользу они будут доставлены на место. Никто даже не заметит ничего подозрительного. Пятнадцать минут роли не играют. А в противном случае они вообще не появятся в зале и конференция будет сорвана. Обвинят потом, конечно, не его, а меня. Я обеспечиваю транспорт, питание, организацию, безопасность. Кракс выходит сухим из воды, я же - теряю пост мэра, общественность отвернется от меня, а уж потом в долгу не останется и Кракс. Понимаешь? - Мэр посмотрел в глаза Джо Блэйда, ища сочувствия.

Джо молчал.

- Я послал пять лучших агентов: Джаббу, Симона, Криса, Кевина и Колина для освобождения заложников, но они вернулись ни с чем. Кракс усилил охрану, и тебе, парень, будет нелегко. Теперь ты знаешь все. Я верю, ты поможешь нам, тем более, что твои крепкие кулаки и меткий глаз теперь найдут настоящую работу. - Мэр похлопал Джо по плечу. - Какое у тебя оружие?

Джо, не говоря ни слова, вытащил из-за пазухи блестящее короткоствольное ружье.

- Я видел эту штуковину у Кракса на заводе, - сказал мэр.

- Да, - подтвердил Джо, - это их последняя разработка - электронный распылитель. Оставляет от человека легкое облачко дыма.

- Причем, опасное, - добавил мэр.

- Пока не рассеется, - кивнул Джо.

- Откуда это у тебя?

Джо повертел ружье в руках и спрятал обратно.

- Купил по случаю в одной лавке.

- Пользуешься услугами Кракса, - вздохнул мэр.

- Исключительно против его системы, - ответил Джо Блэйд и посмотрел в окно.

Утренний туман еще не растаял, да и день обещал быть серым и скучным. Но только не для него. Один, против целой банды головорезов, обученных и хорошо вооруженных. Схватка не на жизнь, а на смерть. И это его устраивало. Он считал возможным помочь мэру, да и всему городу, избавиться от хитрого и страшного Кракса Бладфингера.

- Самое главное это заложники, - сказал мэр. - С Краксом мы успеем разобраться. Для заложников ты - сотрудник безопасности. Чтобы снять их подозрения о каше, которая здесь заварилась, скажешь им что нибудь о взрыве боеприпасов, который вот-вот должен произойти. Наши машины будут ждать недалеко от входа. Помни, что заложники ничего не

знают и чувствуют себя достаточно спокойно. Когда все шестеро будут за пределами базы, найди способ взорвать эту "богадельню", ну а сам сматывай удочки. У тебя будет несколько минут, чтобы унести ноги.

- Ровно двадцать минут, - сказал Джо Блэйд и пояснил: - бомбы Кракса последней разработки срабатывают через двадцать минут после запуска часового механизма.

- Сумеешь разобраться в коде?

Джо кивнул.

- У меня будет тридцать секунд до того, как система самоуничтожения сработает. Этого вполне достаточно, чтобы выставить нужную комбинацию.

Мэр с восхищением посмотрел на собеседника. Он понял, что этот парень знает толк в оружии.

- Запасные патроны есть? - на всякий случай поинтересовался он.

Джо Блэйд отрицательно покачал головой.

- Это только сапожник без сапог, а у Кракса всегда можно найти необходимое. Я надеюсь, что мне хватит двадцати патронов на первое время.

- Ну, смотри сам, - нахмурился мэр и, пристально вглядываясь в даль по ходу шоссе, тронул шофера за плечо. - Тормози, дальше опасно.

Следом за ним остановилась, взвизгнув тормозами, машина черного цвета. Блэйд нащупал ствол.

- Моя охрана, - успокоил его мэр.

Они находились почти за километр от серых кирпичных стен тюрьмы, расположенной чуть в стороне от шоссе на небольшом возвышении. Джо Блэйд осмотрелся. Ни души. Запретная зона. А по сути - одно из владений Кракса Бладфингера. Мэр и Блэйд вышли из машины.

- Советую пройти тем леском, - мэр вздохнул и добавил: - ну, успеха... Я верю в тебя.

Они пожали друг другу руки. Чуть пригнувшись, мэр направился ко второй машине. Хлопнула дверца, и машины почти одновременно рванули с места.

II

Напевая про себя простенькую мелодию, Джо быстрым шагом пошел по направлению к тюрьме и через несколько минут оказался у высокого забора. Вход был чуть в стороне, справа, но Джо рассчитывал на внезапность. Недолго думая, Блэйд перелез через забор. Он оказался на дорожке, ведущей к одноэтажному кирпичному зданию. Стоявший у входа охранник не сразу сориентировался и не успел выстрелить: Джо Блэйд опередил его. "С почином", - подумал Джо и тут же увидел ключ, видимо выпавший из рук охранника. Он поднял его, подождал пока растворится в воздухе белое облачко и направился к двери. Самое трудное было впереди.

Он оказался в длинном, тускло освещенном коридоре. Здесь никого не было. Крепко сжимая в руках распылитель, Джо быстро шел вперед. Он обратил внимание на странные, в человеческий рост витражи, которые украшали стены. Рельефный металлический, устрашающего вида полу-человек, полу-вампир лязгал клыкастой челюстью. "Бред какой-то", - подумал Джо Блэйд. В конце коридора он заметил две человеческие фигуры. Блестящая каска и армейский мундир сразу бросились в глаза. Это был охранник. Второго он не успел разглядеть. Джо выстрелил в охранника. Когда облако рассеялось, он подбежал ближе. В неярком свете Джо увидел человека в строгом сером костюме с надвинутой на глаза шляпой. Заложив руки за голову, он сидел, прислонившись спиной к каким то ящикам. Похоже, заложник даже не обратил внимания на неожиданное исчезновение охранника и появление человека в куртке, кепке и джинсах. Свидание было коротким. Джо Блэйд представился сотрудником безопасности, кратко и не совсем внятно объяснил ситуацию так, как просил мэр.

- А мы ждем транспорт, - развел руками делегат конференции и робко улыбнулся. - Только я отстал и не знаю, где остальные.

- К выходу, там вас ждут! - почти выкрикнул Джо, уже открывая дверь в соседний коридор.

Сразу два охранника, чуть замешкавшись, бросились к нему. Дверь захлопнулась в ту же секунду и он понял, что обратной дороги нет. Джо уже не успевал уворачиваться от здоровенных детин в армейских мундирах. Дав задний ход, он прижался спиной к холодной стене и выстрелил прямо в упор. Дым рассеялся не сразу. Джо, впечатавшись в стену, ждал, пока воздух очистится от молекул распыленных тел. Однако, как он ни старался укрыться от едкого дыма, нескольких вздохов хватило для того, чтобы закружилась голова, а в ногах появилась слабость. В этот момент он пожалел о том, что не взял обыкновенный "Узи", хотя "Узи" все равно выдал бы его шумом.

На стене висел ключ от тюремной камеры. Джо снял его с крючка и пробежал до конца коридора. Здесь было сразу две двери: одна налево, чуть приоткрытая, другая направо - наглухо запертая тяжелая стальная дверь. Джо вставил ключ в замочную скважину стальной двери и, распахнув ее, шагнул в ярко освещенный коридор голубоватого цвета. Там расхаживал охранник с автоматом наперевес. В секунду его не стало. Коридор был свободен. Чувствуя легкое недомогание после прошлой газовой атаки, Джо открыл первую попавшуюся дверь. Это был склад. Он ещо никогда не видел такого количества боеприпасов. Территория тюрьмы служила надежным хранилищем для оружия мафии Кракса. Огромные ящики, стоявшие друг на друге вдоль стен, оставляли лишь небольшой проход по центру. "Да ведь они никогда не будут стрелять здесь", - мелькнуло у Джо. Это было бы самоубийством. Столь простой и логичный довод прибавил ему сил. Он уверенно пошел вперед, внимательно осматривая склад. Неожиданно из-за горы ящиков на Джо бросился толстый улыбающийся кладовщик. В его руке сверкнул нож. Прежде, чем Джо успел выстрелить, лезвие задело висок. Кладовщик метил в лицо, но не попал. Он замахнулся еще раз. Распылитель хлопнул в то мнгновение, когда, казалось, длинный кривой нож, как сабля разрубит тело пополам. Но Джо успел.

Пока растворялось облако, Джо переодевался в мундир охранника, найденный среди ящиков с боеприпасами. Тяжелая каска, сапоги, материал, пропитанный специальным составом, пожалуй, могли защитить и от опасных молекул распылённых тел, а уж о том, что в таком облачении Джо Блэйд ничем не отличался от всех остальных охранников, говорить не приходилось. Еще раз внимательно окинув взглядом склад, Джо побежал к выходу.

Двадцати патронов могло не хватить, и Джо это понимал. Он надеялся, что рано или поздно найдет патроны нужной системы. Однако среди этого военного великолепия ничего подходящего он не видел.

Тюрьма поражала своими размерами и Джо Блэйд подумал о плане помещений. Конечно, было бы не плохо его иметь, но, как всегда, приходилось рассчитывать только на себя. К тому же нужно было экономить время, патроны и силы. Пару раз Джо удалось проскочить незамеченным. Но даже когда он лоб в лоб столкнулся с охраной одной из камер, никто не обратил на него внимания. Его спасал мундир, найденный на складе.

Вскоре он нашел бомбу. Огромный шар черного цвета лежал прямо в проходе. Джо хотел оставить ее незаряженной, но, перепрыгивая через бомбу услышал щелчок, сигнал зуммера, и понял, что жить ему осталось ровно тридцать секунд. Высокочувствительный механизм срабатывал от малейшего движения. Код состоял из пяти букв и нужно было выставить определенную комбинацию. Шифр Джо знал. Один из продавцов оружия рассказал как-то о новой разработке военных - супербомбе, и сказал, что пока идут опытные образцы. Шифр прост - A B C D E. Именно в такой последовательности следовало выставить буквы. Сложность заключалась только в том, что передвигать буквы по табло можно было лишь по заданной схеме. Таймер безжалостно отсчитывал секунды. Когда до самоликвидации бомбы оставалось пять секунд, Джо установил последнюю букву на место и услышал длинный монотонный сигнал. На счетчике засветились цифры 20.00 и начался обратный отсчет времени. Теперь остановить таймер не смог бы и Господь Бог.

Почти одновременно с сигналом неожиданно выстрелило ружье Джо Блэйда. "Что за черт?" - подумал Джо. На один патрон, действительно, стало меньше и Джо понял, что сильный электромагнит в бомбе заставил сработать электронный боек его ружья. "Такими темпами у меня через пять минут будет пустой магазин". Но Джо решил, что лучше пожертвовать патронами, чем опоздать, и не стал особо переживать. Ему нужно было найти

еще пятерых заложников, да и одной заряженной бомбы на тюрьму явно не хватит.

В следующем помещении Джо понял, что от военной формы придется отказаться. Здесь охранники были одеты в ту же форму, но другого цвета. Он не рискнул нарушить порядок и в надежде на новую форму стянул с себя старый мундир. Пробежав через коридор и стараясь быть незамеченным, Джо нырнул в открытую дверь. Тюремщик с большим животом видимо собирался вкусно поесть. На столе стояла бутылка воды, в широкой вазе лежали фрукты. Джо не был голоден, но знал, что при молекулярном отравления лучше всего помогают именно вода и фрукты. Тюремщик не успел даже вскочить из-за стола. "Ради этого стоило потратить патрон", - подумал Джо, доедая яблоко и запивая его водой. На часы он не смотрел. Лишняя суета - плохой союзник. Подкрепившись, он сразу почувствовал себя лучше.

Он плохо помнил, как в течение двух минут успел найти и зарядить еще две бомбы, а также освободить ничего не подозревающего делегата.

В длинных коридорах тюрьмы он редко стрелял, предпочитая уворачиваться от ударов и пробегать мимо охранников там, где это было возможно, зная, что ни один не решится выстрелить в набитой до отказа боеприпасами тюрьме, Джо просто прыгал как пантера через ошалевших бойцов и рвался вперед.

Он еще раз нашел новую, с иголочки, форму и уже в ней, давая себе передышку, вышел на улицу во внутренний двор тюрьмы. Теперь стало ясно, что заложников развели по разным зданиям и коридорам. Каждый, видимо, думал, что отбился от основной группы. Охрана относилась к ним лояльно и поэтому бунта не возникало. Джо посмотрел на пасмурное небо и подумал о своем возвращении. Бежать из тюрьмы можно было только через выход, минуя коридоры и охрану. Дорогу назад он помнил. В одной из камер он оставил нетронутыми еще один мундир и каску, которые позволят уйти незаметно.

Только сейчас он рассмотрел на часы. До взрыва оставалось одиннадцать минут.

Убрав охранника возле небольшого кирпичного дома, Джо с радостью обнаружил внутри боекомплект для своего оружия. Десять патронов, тщательно упакованные в небольшом ящике, быстро перекочевали в магазин распылителя. На душе стало легче. В соседней комнате он нашел безмятежно спавшего мирового лидера. Джо объяснил ситуацию. Тот быстро согласился и направился бодрым шагом к выходу. Джо знал, что этот серый костюм никто не тронет и он спокойно найдет выход и всех остальных. Спустя несколько секунд Джо чуть было не поплатился жизнью, оказавшись один на один в тесной камере с каким-то заключенным. Вплотную прижав Джо к стене, он принялся душить его. Схватка была короткой, но решительной. Силы уже покидали Джо Блэйда, когда ему удалось, оттолкнувшись от стены, перепрыгнуть за спину противника и, быстро развернувшись, выстрелить тому в живот.

Сквозь облако испарений Джо увидел, что входная дверь захлопнулась и он оказался в каменном гробу из четырех стен. Пот лился градом, а Джо лихорадочно шарил по карманам. Наконец ключ был найден. Слава Богу, что он зашел сюда с запасным ключом, иначе пришлось бы ему последние десять минут жизни провести за тюремными стенами. Джо снова вышел во внутренний двор. Слева от него был целый ряд самых разных построек, но, чтобы проникнуть туда, нужно было пройти мимо центрального поста, а потом уж повернуть налево. Бдительный охранник получил свою порцию смертоносного луча, когда пытался задержать Джо у входа. Блэйд подошел к дверям. Их было две. Он открыл правую и на этот раз интуиция не подвела его: в одной из трех тесных комнат Джо обнаружил комплект боеприпасов для распылителя, немного еды и уставшего заложника. Полдела сделано. Ему осталось осмотреть ту часть построек, которые находились слева от центрального поста. Дверь еще не успела открыться, как в лицо Джо выдохнул перегаром стоявший напротив охранник.

- Стой! Кто такой?

Вопрос в лоб. Ответ был достоин вопроса. За Джо заговорило его ружье. Легкое облако быстро разносил ветер. "Интересно", - подумал Джо, - "они уже ищут меня или нет?" Судя по всему, информация о проникшем на территорию тюрьмы постороннем не поступала. Джо застал всех врасплох. Чуть больше десяти минут находится он здесь, а

кажется, что прошла целая вечность.

У входа в первое здание, слева от главной дороги, сразу два солдата угрожающе шагнули к Джо. Он выстрелил. Чуть подумав, решил не идти через дверь, а влез в окно.

Длинный зеленый коридор показался ему сплошным кошмаром. Он изрядно потратил патроны, прежде чем добрался до черной двери. Здесь он нашел сразу несколько ключей: они обеспечивали свободу перемещения и были нужны, как воздух. Три ключа Джо оставил в запасе для возвращения обратно. Вскоре он нашел еще двух заложников, отдыхавших по разным концам коридора. После прощания с ними Джо в темпе осмотрел несколько камер. В одной из них он нашел бомбу и на этот раз запустил ее без труда. Минуты летели с неимоверной быстротой. Шесть кругов оставалось пройти секундной стрелке до взрыва, а он еще не нашел последнего, шестого делегата конференции. Эх, знать бы, где он!

А заложник был недалеко, в соседнем здании. Джо не стал ничего объяснять. Показав ему дорогу к выходу, он помчался по коридору в надежде найти патроны. Патронов он не нашел, а за две минуты успел зарядить еще две бомбы. "Подарок Краксу", - подумал Джо, - "шесть бомб - шесть заложников. Меняемся!", - он улыбнулся своим мыслям, однако спустя несколько секунд понял, что радоваться рано. Возвращаясь к центральному посту, Джо наткнулся на вновь прибывшую смену. Не раздумывая, он нажал на курок, но ружье молчало. Патроны кончились, Джо повернул назад. К нему шли еще двое. В отчаянии он применил свой коронный прием: прыжок с выброшенной вперед ногой заставил чуть наклониться охранника и Джо буквально перелетел через него. Тот бросился сзади, но не успел - Блэйд бегал быстрее.

Три ключа были в кармане. Где-то в глубине самого первого здания его дожидался спасительный мундир. Нужно было только успеть. Теперь, видимо, информация о нем дошла и его, наверняка, попытаются либо убить, либо взять живым. Выйти из тюрьмы ему не дадут…

…За две минуты до взрыва Джо в форме охранника бежал по коридору к выходу. На него не обращали внимания. Все искали коренастого в штатском.

Последний ключ он вставил в дверь за полторы минуты до взрыва. Дорожка с которой он начинал свою миссию, теперь показалась ему слишком длинной. Но все! Он успевает!

Черный лимузин увозит его прочь от обреченной тюрьмы. В машине, кроме шофера, он один. Все остальные готовятся к конференции и лишь немногие знают, что через минуту тюрьма Кракса перестанет существовать.

Все остальное будет потом: награда, всеобщее признание, свобода, наконец. А в эту минуту Джо Блэйду хотелось одного: откинувшись на спинку сиденья, вдохнуть глубоко воздух и дождаться красного зарева и раскатов грозы над тюрьмой. Джо все еще видел в зеркало заднего вида очертания тюремных стен и построек. Он молча ждал. Шофер тоже молчал. Только мягкое гудение двигателя да шорох шин нарушали тишину.

СОВЕТЫ ЭКСПЕРТОВ

MYTH. History in the making

Автор: Neil Dodwell, 1989
Фирма: THE SYSTEM THREE

Эксперт Матыцын Д.А.
г. Москва

Эта игра является довольно развитым представителем жанра аркадных адвентюр и требует от играющего как быстрой реакции и сообразительности, так и логического мышления.

Фабула игры основана на том, что по неизвестным причинам нарушилась ткань времени и в нашу эпоху стали проникать мифические создания и герои древних легенд. Для исследования причины такого смешения Вам поручается провести историческую экспедицию и побывать в некоторых узловых точках истории человечества.

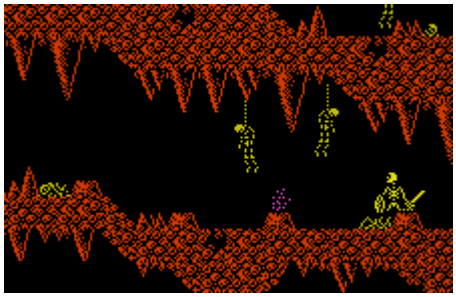
Игра многоуровневая, с пятью дозагрузочными этапами. Для перехода с одного уровня на другой, необходимо собрать пять энергетических шаров. Первые четыре из них обычно находятся довольно легко, последний же, как правило, хитроумно запрятан или защищается могучим стражем.

В верхней части экрана индицируется количество оставшихся попыток и очки (слева), жизненная энергия героя и количество собранных шаров (справа), а также задействованный в данное время предмет (в центре). Когда герой теряет более 80% энергии, его изображение начинает мерцать.

В начале каждого уровня дается обычно один предмет, используя который Вы добываете и другие. Смена предмета - нажатием клавиши SPACE. Кнопка "огонь" совместно с "вправо" или "влево" позволяет использовать текущий предмет.

I
"The Road to Hell".
("Дорога в ад")

Итак, Вы в далеком прошлом. Ваш маршрут начинается под старым кладбищем. Единственное исходное оружие - удары ногой или рукой. Они выполняется нажатием клавиш "вправо", "влево" или "вверх" при нажатой клавише "огонь". Ногой открываются сундуки и разбиваются сосуды, внутри которых могут быть найдены некоторые полезные атрибуты, восстанавливающие энергию героя, дающие временную неуязвимость, дополнительные заряды и т.п.



Основные опасности - скелеты, монстры, извергающие огонь, смертоносные вулканы. Первые три энергетических шара собрать легко - найдя шар, надо выстрелить в прыжке. Шар упадет и его можно взять. С остальными шарами дело обстоит сложнее...

Расстреляв не менее десяти ходячих скелетов, подберите оставшиеся от них черепа, спуститесь на самый нижний этаж и идите влево к огненному озеру. Там бросьте эти черепа в пламенные воды. Из пламени возникнет демон. Победив его, Вы сможете по возникшему островку добраться и до четвертого энергетического шара. На этом же островке надо взять мощное оружие - трезубец и идти к другому огненному озеру. Из середины озера поднимется островок, который позволит дойти до гигантского огнедышащего дракона, победить которого Вам поможет волшебный трезубец, а боевым трофеем станет ключ, которым открывается дверь пещеры, в которой хранится последний шар.

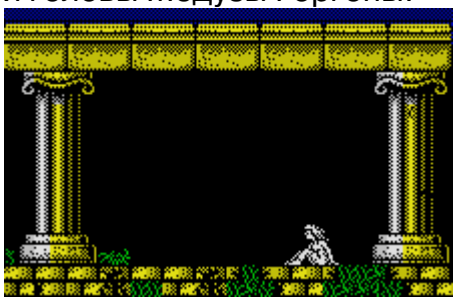
Последняя задача - переход на новый уровень. Портативный телепортер Вы найдете на нижнем этаже, неподалеку от огненных озер. А использовать его надо там, откуда Вы начинали игру (место отмечено воротами, на которых есть изображение Вашего телепортера).

II

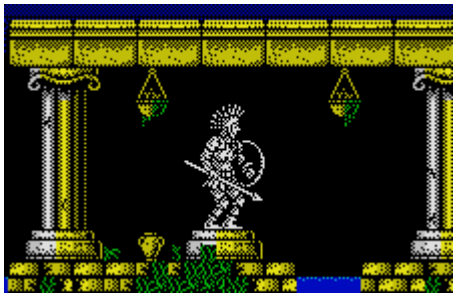
"Greece. Four Hundred BC".

(Греция. 400 лет до н. э.)

Пронизывая пласты времени, Вы попадаете в 400 год до н.э. И вот перед Вами древняя Греция. Свой первый шар Вы найдете без труда, а уничтожив мечом пару колонн, подберете спрятанные за ними атрибуты и волшебный мешок. Он пригодится впоследствии для транспортировки смертоносной головы Медузы Горгоны.



Если Вы сумеете успешно отразить атаки агрессивных привидений и минуете статую льва, то попадете в уютный дворик, в конце которого отдыхает прекрасная нимфа. Однако, при Вашем приближении она превратится в огнедышащего монстра, единственное спасение от которого - спрятаться за колонну. Вдоволь натешившись, нимфа улетит, а Вы станете обладателем еще двух энергетических шаров.



С опаской поглядывая наверх, откуда начал капать смертельно ядовитый дождь, подойдите к пустому портику. В него можно войти и передохнуть, ненадолго присев.

Ваша следующая задача - Медуза Горгона, охраняющая четвертый шар. Хорошо, что у нас есть бронзовый зеркальный щит - им можно прикрыться от ее испепеляющего взгляда. Когда возьмете шар, заодно отрубите Горгоне ее голову и осторожно положите ее в свою волшебную сумку.

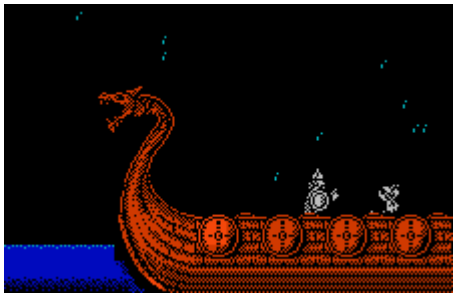
Если на обратном пути хорошо обыскать пещеры, Вы непременно найдете огромного трехголового дракона. Быстро достаем голову медузы Горгоны и с ее помощью уничтожаем одну за одной все три головы страшной рептилии. Победив дракона и забрав пятый шар, Вы без труда вернетесь за телепортером в пещеру Горгоны, возвратитесь к месту старта и, немного отдохнув, продолжите свою экспедицию.

III

"Scandinavia. Five Hundred AD"

(Скандинавия. 500 год н.э.)

Теперь мы попадаем к норманнам в Древнюю Скандинавию. Очутившись на корабле Драккаре в разгар свирепого шторма, сопровождающемся громом и молниями, Вы не сразу сообразите, что от Вас требуется.



Осмотритесь, опробуйте свой огромный топор и можете приступать к битве с командой корабля. Если сражение пройдет нормально и Вы победите главного викинга - конунга, то взяв его оружие, Вы сможете приобрести первый энергетический шар, после чего окажетесь уже на суше.

Здесь Вас ждет толпа злобных гоблинов, вооруженных дубинами. Трофеями этой победы будут ножи, которые впоследствии Вам пригодятся. Изредка вашим взорам будут представать прекрасные лесные феи - брумгильды, но они не менее опасны, чем гоблины. Идя налево и разбивая по пути ящики и сосуды, Вы найдете второй энергетический шар, после чего можно начать исследовать правый фланг.

Погасить жертвенный костер Вы сможете, вызвав дождь с помощью заклинания из старинной рукописи. На месте пепелища Вы найдете ключ от замка скандинавского бога Одина.

Двигаясь направо, Вы придете к третьему шару, но дальнейший путь преградит злобный дракон Нидхог, справиться с которым Вы сможете только метанием ранее собранных ножей с достаточно близкого расстояния.

Четвертый шар Вы найдете на подступах к замку Одина. Попасть в замок Вы сможете, опустив подъемный мост через ров, воспользовавшись ключом из жертвенника. В замке Вас встречает сам бог Один. Есть только один предмет, которым можно его поразить. Интересно, успели Вы найти к этому моменту молнию? Если да, то победа и пятый шар будут Вашими.

Телепортер найдете на обратном пути, а пользоваться им Вы уже умеете.

IV
"Egypt Three Thousand BC".
(Египет. 3000 лет до н. э.)

На этот раз Вас забросили в историю еще дальше. На календаре трехтысячный год до нашей эры. Древний Египет. Выбравшись из великой реки Нил, подойдите к пирамиде. Не правда ли, войти туда не удается? Как будто неведомая сила отталкивает Вас от входа. К счастью, в карманах завалялся револьвер и пары выстрелов достаточно, чтобы разнести дверь в щепки. Теперь путь свободен! Мы попадаем в пирамиду и спускаемся вниз.

Главный мотив этого уровня лабиринт. Отсюда пути ведут в разные стороны. Слева - четыре ниши, в каждой из которых начинается своя часть лабиринта, а справа от развилки - пятая, которая откроется после прохождения первых четырех ветвей. На каждой ветви Вы найдете по одному энергетическому шару и по одному древнему предмету - артефакту.

Начнем свое исследование с самого левого хода. Сразу подберем первый шар. Уворачиваясь от острозаточенных маятников и вскрывая по пути сундуки и сосуды

доберемся до первого артефакта.

Вернувшись назад, пойдем во вторую ветвь и бежим направо. Здесь по шару надо выстрелить издалека. Близко не дайдут подойти острые шипы, торчащие из пола. Впрочем, после того, как шар упадет, шипы задвинутся обратно. Забираем его и в левом тупике второй ветви берем второй артефакт.

В третьей ветви у Вас есть интересная возможность восстановить все жизни своего героя. Это делается с помощью магического креста. Надо использовать его, стоя под изображением аналогичного креста. Стоит время от времени приходить сюда на "подзарядку".

Как и в прочих ветвях, здесь Вы также найдете энергетический шар и артефакт.

Четвертая ветвь довольно проста, стоит опасаться только шипов в полу. Берем шар и артефакт, после чего возвратаемся на развилку.

Вы увидите, что изображение глаза в пятом проходе начало мерцать. Вставьте на это место и используйте такой же предмет, как и глаз - попадете в начало длинного туннеля. После тяжелой борьбы Вам удастся прорваться к золотому саркофагу. Здесь надо выложить все четыре артефакта и тогда Вы встретите основного противника - огромную золотую маску Рамзеса. Победить ее можно с помощью припасенной ранее головы фараона.

Дальше все просто: взять последний шар, найти телепортер и, собравшись с духом, переходить на следующий уровень.

Примечание: на четвертом уровне следует как можно чаще передвигаться прыжками, а не бегом. Здесь много мест с падающими стенами, с шипами в полу, с острыми секирами, которые при первом же прикосновении начинают с угрожающей быстротой раскачиваться, как маятники.

V

"Duel with Dameron" ("Дуэль с Дамероном")

Все дальше и дальше мы проникаем вглубь времени. Уже не видно и людей, вот и Земля куда-то пропала. Мы летим в бесконечном космосе. Неожиданно перед нами поднимается злой демон времени - ангел Хаоса Дамерон. Теперь понятно, кто виновен во всех неполадках. Ничего не остается, как сразиться с ним. Победить его непросто. Уворачиваясь от летящих в Вас огненных шаров и тщательно прицеливаясь, Вы поражаете Дамерона в единственно уязвимое место - туда, откуда вылетают его шары.

Но он не погиб. Потеряв значительный кусок своего тела, он перестраивается и продолжает извергать смертельные заряды. Снова и снова отбиваете Вы от него кусок за куском и, наконец, враг повержен.

Так заканчивается Ваша историческая миссия. Ткань времени восстановлена, цивилизация спасена!

THE DARK WHEEL

Продолжение. Начало см. в ZX РЕВЮ-92, стр. 175-176, 218-220, 262-264.

ГЛАВА 5

Выход из подпространства как всегда сопровождался тошнотой, потерей ориентации и головокружением. Легкая дрожь охватила тело. Прямо перед ними, на приличном расстоянии тускло светил красно-голубой диск планеты Ксезавр. Он не очень выделялся на фоне сверкающих звездных полей. Тусклое солнце этой системы находилось неподалеку. Умирающая звезда с трудом проливает красный свет на этот застывающий мир. И никакая индустриальная и научная мощь планеты не в силах остановить этот неумолимый процесс старения галактики.

Да, Ксезавр превратился в мир, в котором уют и тепло ценятся превыше всех благ на свете. Вот где они получат достойную цену за те шанаскилские меха, которыми загружен их корабль.

Как обычно, все шло по раз и навсегда установленному порядку. Подходит к концу еще один скучный торговый рейс. Элиссия сладко зевнула, а Алекс ввел замеренные координаты в автопилот и приготовился убить время, необходимое для скучного подхода к планете.

Однообразный, установленный порядок, к которому Алекс уже давно привык: выход из подпространства, медленный перелет, швартовка у станции "Кориолис", - и нечего делать, и не на что смотреть.

Вдруг "Кобра" вздрогнула, эхом в боевой рубке отозвался скрежет разрываемого металла.

- У нас компания, - вскрикнул Алекс.

Элиссия очнулась от сна и захлопала удивленными глазами, стряхивая остатки дремоты. Она поняла ситуацию в мгновение ока, но осталась неподвижной в своем кресле. Алекс был у консоли и уже владел ситуацией.

В какой-то степени его поймали врасплох. И дело вовсе не в элементарной невнимательности, дело в том, что атакующие корабли оказались необычно близко к точке выхода из гипертуннеля. Более того, они занимали положение между объектом своей атаки и солнцем, так что какое-то время оставались невидимыми и имитировали при этом движение астероидов, беспорядочно кувыркаясь и дрейфуя в пространстве.

Алекс в общем-то их видел, но проигнорировал и теперь его "Кобра" приняла на себя первый залп.

Противники сгруппировались сзади и образовали боевой порядок. Непрерывно сканируя пространство, Алекс набирал максимальную скорость.

Вновь заскрежетали энергетические щиты, на которых заиграло лазерное пламя очередного залпа.

- Лучевые лазеры! Да, эти корабли неплохо вооружены. - подумал Алекс, - но и наша "Немезида" теперь им не уступит!

"Немезида" - такое драматическое имя древней богини мести дали Алекс и Элиссия своему кораблю.

Алекс включил экран заднего обзора, подогнал прицельный квадрат и дал две коротких вспышки из новехонького кормового лазера.

Корабли противников сломали строй и разошлись. Один из них закувыркался. Пока враг был на экране, Алекс нацелил ракету.

О приближении вражеской ракеты оповестил экран, вспыхнув сообщением об опасности, впрочем Алекс видел ее и так. Он немедленно привел в действие систему ECM и после томительных секунд ожидания ракета исчезла в жаре и пламени.

Тряхнуло корпус и Алекс нырнул вниз. Краем глаза он успел заметить сброс энергии

защитных экранов в первый энергетический отсек.

Элиссия спокойно сидела в своем кресле. Алекс уверенно владел ситуацией. Прямо перед ними стремительно нарастал диск планеты. Казалось, что он то взлетал, то падал, вращаясь при этом с головокружительной скоростью - это Алекс маневрировал, занимая наилучшую боевую позицию.

Он действовал почти инстинктивно. Резко развернув свой корабль, он вдруг решительно бросил его навстречу ближайшему преследователю. Это оказался "Фер-де-Ленс", быстрый и изящный корабль, по-видимому сверху донизу забитый самым изощренным навигационным и боевым оборудованием, установленным его прежним владельцем. Впрочем, может быть и нет... Для содержания такого оборудования необходима куча денег, а этот корабль уже многое повидал на своем веку.

Сближаясь с пиратом, Алекс решил применить ракету. У них на борту было четыре ракеты, причем одна из них уже была нацелена. Он нажал на пусковую клавишу, "Кобра" вздрогнула, смертоносный снаряд пошел к своей цели. Она нашла ее и "Фер-де-Ленс" буквально исчез.

"Может ушел в гиперпереход?" - подумал Алекс, - "Вроде не похоже". На самом деле он проскочил мимо противника и теперь, включив задний экран, увидел огромное отливающее серебром облако, быстро расширяющееся навстречу звездам.

- Отличная стрельба! - вдохновенно откликнулась Элиссия.

Из облака пепла и металла проступили контуры другого корабля и Алекс вновь ушел в петлю. Новый лазерный заряд принял на себя кормовой экран. Теперь, когда охотник знал, что его жертва обладает системой подавления ракет, он решил "сесть на хвост" и добивать ее лазером.

Это тоже была "Кобра". Грузоприемники широко распахнуты, готовые подобрать контейнеры с драгоценный шанаскидским мехом разбитого торговца.

Но у Алекса на сей счет были иные планы. Маневр. Ксезавр опять перед ними. Огонь кормового лазера. Нырок вниз, разворот, выход в непростреливаемую зону и Алекс нацелил новую ракету.

- Не надо, побереги ее, - с сожалением вздохнула Элиссия.

- Я знаю, но нам хватит денег на замену, - сказал Алекс.

- Тогда не хватит на энергозаборники, - напомнила Элиссия и оба от души расхохотались. В такое время они думали о новых покупках!

До станции с ее системой обеспечения безопасности оставалось еще слишком далеко. Алекс сделал крутой вираж к солнцу и резко сбросил скорость. Преследователь идеально повторил первый маневр, но на несколько секунд запоздал с ориентацией и не успел затормозить. Он не успел еще понять, что произошло, а охотник и жертва уже поменялись местами.

- Давай, Алекс, давай! - кричала в восторге Элиссия, а Алекс вгонял один за другим лазерные заряды в тело вражеского корабля. "Кобра" на экране крутилась и изворачивалась, но Алекс не думал, он только реагировал и методично добивал противника. Температура носового лазера опасно повысилась. Пират выпустил ракету, но Алекс не стал отвлекаться на программирование ECM и спокойно расстрелял ее лазером.

У Элиссии перехватило дыхание и она в восторге смотрела на юношу, уверенно державшего в руках их жизнь и судьбу.

Еще через мгновенье все кончилось. Пират взорвался, энергии его защитных полей не удалось сдержать такой натиск. Алекс успел увидеть мелькание спасательной капсулы и на мгновенье вспомнил ту вспышку огня, которая уничтожила его собственный корабль, он вспомнил гибель родной "Авалонии".

Его первой реакцией было броситься в погоню, но подумав, он успокоился и отказался от этой идеи. А вокруг сверкали и кувыркались контейнеры с разбитого корабля.

- Подумать только, какая жалость, что у нас до сих пор нет грузозаборника! – пробурчала Элиссия.

Зато мы получим премию за пиратов, - ухмыльнулся Алекс, - и притом за двоих.

- Алекс, ты был великолепен. Летать с тобой между звезд - большая честь.

Алекс спокойно вел свой корабль к планете и больше никто не сказал ни слова. Они даже не упомянули тот факт, что это было первое настоящее сражение, которое он провел без ее помощи.

ГЛАВА 6

Они уже торговали в течение трех месяцев и теперь в "Немезиде" трудно было бы узнать ту "Кобру", которая когда-то была надгробным камнем Генри Белла. Покрашенная, расцвеченная опознавательными знаками, оснащенная новыми боевыми надстройками, она все более и более походила на боевой корабль.

Три месяца торговли и ни на час Алекс не забывал о главной цели своей жизни. Кто-то или что-то, замаскировавшись под мирного торговца, убил его отца и сделал это преднамеренно. Да, его отец вел сложную жизнь и, согласно неписанной галактической традиции, сделал все возможное, направив и сына по тому же пути. И Алекс Райдер оправдает веру отца.

Вопросов перед Алексом возникало не меньше, чем боли и гнева. То же состояние переживала и Элиссия, хотя она, как и все женщины Теорга, редко проявляла свои эмоции. Тем не менее, Алекс чувствовал кипение горячих чувств под холодной внешнее оболочкой.

У них была общая задача. Они должны были выжить, стать сильными и отомстить. Им предстояло долго вместе ждать и оба понимали, что лучшего и более молчаливого партнера для своей цели им не найти. В то же время, им обоим было нелегко…

Встреча с пиратами немного повредила кораблю. Здесь и там пострадала краска, разболтались некоторые панели облицовки боевых надстроек. Это означало необходимость посещения станции обслуживания. Впрочем, этой возможностью стоило воспользоваться, поскольку им, как победителям пиратов, этот сервис полагался бесплатно.

Это сражение для Алекса было первым самостоятельно выигранным боем, но это был не первый их бой. Если бы Элиссия не скрывалась, она вполне имела бы право подавать заявку на ранг "Опасный". А так как ей приходилось сохранять свое инкогнито, все победы заносились в реестр Алекса. Сегодняшний бой Алекс воспринял как значительный шаг вперед и впервые у него появилось чувство, что он по-праву носит тот ранг, который ему присвоили благодаря успехам Элиссии.

Он направил свой корабль внутрь тысячекилометровой зоны над поверхностью планеты и ее диск заполнил собой весь экран переднего обзора. Корабль развернулся на самой малой скорости и медленно вращающийся металлический куб засверкал перед ними - планетная станция с распахнутым зевом приемного отсека.

- Кстати о стыковочном компьютере.., - начал было Алекс, уравнивая скорости вращения корабля и станции.

- Пустая трата денег... - пресекла его попытку Элиссия. - Если ты не можешь стыковаться так, чтобы не ободрать краску с бортов, тебе вообще нечего делать в космосе.

Алекс был прекрасным пилотом, но вход в приемный отсек станций типа "Кориолис" был его самым слабым пунктом. Он, конечно, сделал это и на сей раз, но вздохнул спокойно только тогда, когда корабль, вошедший в просторный ангар, был подхвачен магнитным манипулятором и притянут к свободному стапелю. Тут же стыковочная штанга автокома скользнула вдоль корпуса и подключилась к разъемам корабля.

Покя Элиссия как обычно пряталась в спасательной капсуле, Алекс с интересом рассматривал кубическое внутреннее пространство станции: таможенные, полицейские, рекламные корабли, ремонтные модули, - все это медленно перемещалось в пространстве. У каждого свое дело, каждый знал свой бизнес.

По каналам автокона Алекс доложил о грузе на борту и получил подтверждение регистрации победы над пиратами и свидетельство начисления призовых тридцати кредов. Это как раз то, что нужно для покупки новой ракеты.

Когда все формальности проверок и перепроверок были закончены, Элиссия наконец покинула свое убежище. Так уж получилось, что именно спасательная капсула стала для них предметом первейшей необходимости, без которого они не могли бы посетить ни один порт. Им удачно удалось купить подержанную всего за четыре сотни и само собой

разумеется, что они не собирались использовать ее по назначению.

Теперь начались обычные торговые процедуры. Сначала продажа, затем обсуждение очередного рейса и, наконец, покупка нового товара.

Торговля во многом напоминает игру. При известном спросе и быстром обороте всегда можно получить небольшой, но надежный доход на продуктах, текстиле, простейшем промышленном оборудовании и незамысловатых безделушках для богатых. Но непрерывно возрастающая стоимость обслуживания корабля и потери от периодических стычек очень скоро съедают эту скромную, с таким трудом полученную прибыль, после чего все предприятие становится практически бессмысленным. Узнать цену на те или иные товары в другой звездной системе нет никакой возможности. Все правительства ревниво оберегают коммерческую информацию о своих рынках. За передачу информации о ценах по каналам факс-связи следует очень тяжелое наказание.

Естественно, цены нестабильны. Во всех системах, даже в самых бедных, орудуют сети спекулянтов. Еще вчера за тонну пустоpесничника, купленного на Реорте за три креда давали на Цейнзиле по восемь, а сегодня с трудом покупают всего за два. И дело вовсе не в том, что спрос на пустоpесничник сократился. Просто банда спекулянтов сыграла свою игру и подмяла рынок.

До сих пор Алексу и Элиссии везло. Они перевезли варгорнские шелка от Рексебе на Инеру и удвоили свою первую сотню кредов. Потом они торговали золотистой чешуей геретеанских рептилий и с трудом вернули затраченное. Они поставили двадцать тонн семян подсолнечника амфибиоидам на Биерле, для которых это был изысканный деликатес и лишь прибыв к месту назначения с удивлением обнаружили, что всепланетная мутация изменила вкусовые железы обитателей... и поставила последних перед проблемой поиска новых деликатесов. Достаточно близко подходили для этой цели ароматические масла и лавандовая туалетная бумага, но все же это не то. Где-то все-таки должна быть возможность НАСТОЯЩЕГО дохода и рано или поздно он придет.

Переброска оборудования из высокоразвитых миров в среднеразвитые оказалась весьма прибыльным делом. Алекс и Элиссия быстро поняли также, что на развивающихся промышленных планетах всегда можно иметь хороший доход от торговли предметами роскоши, но самой прибыльной обещала стать операция с продажей шанаскилского меха на Ксезавре. Они купили его по тридцать кредов за тонну и сейчас Алекс в нервном напряжении ожидал ответа на запрос биржевых сводок по мехам.

Он не удержался и в триумфальном жесте вскинул руки, когда понял, что они с Эмиссией утроили свой капитал. На этот раз им улыбнулось настоящее счастье.

Мех они продали без проблем и запросили сводку цен на бортовое оборудование и вооружение. Новая ракета стоила свои стандартные тридцать кредов. Алекс заказал одну и автоматический робот приступил к исполнению заказа. Лучевые лазеры по тысяче за штуку являли собой почти непреодолимое искушение. Топливо и совершенно необходимое энергогрузозаборное устройство обходились в пятьсот двадцать пять, а энергетическая бомба стоила еще вдвое больше.

Конечно, энергозаборники обещали большую экономию. Можно будет в дальнейшем не тратиться на горючее, а пополнять баки, подпитываясь от звезд. Это было хорошее вложение, даже при том, что цены здесь были на сотню выше, чем обычно. Алекс заказал одно устройство. На доставку и монтаж уйдет примерно двадцать часов (один стандартный день). Заправив корабль, Алекс приступил к изучению списка Ксезаврианских деликатесов. У них еще оставалось триста двадцать кредов - сумма, с которой чувствовать себя было неуютно. С другой стороны, корабль имел дополнительную энергетическую защиту, лазеры с четырех направлений, полный боекомплект ракет и грузозаборник. Пройдено более половины пути превращения торговой "Кобры" в боевой крейсер.

Элиссия тоже внимательно изучала список. Удивительно, но ксенозаврианцы, большие любители экзотики, мало что могли предложить в этом плане. Так, например, всего два вида наркотиков было на рынке: арктурианский лопнитрав и, как ни странно... табак. Алексу было над чем подумать.

- Думаю, что с табаком можно было бы попробовать...

- Не-а, - промолвила Элиссия. - Не выйдет, никотин смертелен в малых дозах для большинства рас.

- А если в гуманоидную систему?

- Все равно опасно.

Широко предлагались минералы, но цены были не те. Дюрасьон - руда, из которой после очистки и старения получают дюралиум для строительства кораблей, шел всего по восьми за тонну. А он исключительно хорошо продается на Лейве, но до Лейва отсюда столько световых лет! Не подходит.

Драгоценные камни? Предлагались каштанит и серебристый спектонал. Еще были красно-зеленые эмеронды. Но пираты учуют такую добычу даже за пару световых лет.

Чисто из любопытства Элиссия обратила внимание на продажу двухсот окаменевших костей дирнофаксаурина по сорок кредов за штуку.

- Когда нибудь слышал о них?

Алекс ответил:

- Даже видел одну в музее у себя дома. Они живые и поют. Им более сорока миллионов лет и, тем не менее, они поют. Непонятно, чего они ждут. То ли им нужна какая-то наседка, то ли ждут изменения климата. Это тазовые кости, так что возможно, что они служат чем-то вроде инкубационной камеры, но толком никто ничего не знает.

- А они ценятся?

- Очень, хотя точную цену я не знаю.

- Проверь по списку запрещенных грузов.

Алекс проверил. Никаких известных ограничений на импорт этих окаменелостей он не нашел,

- Кажется, это интереснее, чем еда.

- Это уж точно.

- Так берем..?

- Пожалуй...

Но не успел Алекс набрать на клавиатуре заказ торговому центру, как вспыхнул экран "Получено сообщение...".

- Рейф. - радостно сказал Алекс. Элиссии тоже понравилась перспектива повидать Рейфа Зеттера и поговорить с ним еще раз.

Но вовсе не старый космический торговец появился на экране, когда Алекс вышел на связь.

Изображение на экране явно принадлежало человеческому существу, а не гуманоиду. Но его лицо не поддавалось никакому описанию. Есть масса способов изуродовать человека до неузнаваемости и превратить его в подобие ночного кошмара, например пролететь слишком близко от звезды, слишком часто подвергаться воздействию межзвездного вакуума, поработать в некоторых рудниках... Но сейчас, глядя на бугристые серые опухоли, покрывающие плоть человеческого лица, Алекс не мог себе даже представить что произошло с его собеседником.

Губы, как обрывки паутины, затрепетали на серой плоти, скелетообразная скрюченная рука, сквозь которую просвечивали красные кровеносные сосуды, коснулась тонкой пряди волос, причудливо спадавшей с деформированной головы.

- Ты Райдер?

По крайней мере, голос был нормальным и притом мужским.

- Назовите себя.

Игнорируя вопрос, незнакомец продолжал. - Чем сегодня торгуешь? Минералы? Специи?

- А вам зачем?

- Не знаю твоих планов, но что бы ты ни делал, у меня есть для тебя предложение получше.

- Не стал бы с тобой торговать даже ради спасения от сверхновой.

- А Рейф Зеттер бы стал. Чего это ты такой нервный?

- Ты знаешь Рейфа? – удивился Алекс, озадаченный упоминанием знакомого имени.

- И я и пол Вселенной. - Изуродованный человек наклонился вперед. Черты лица заняли весь экран. Это паразиты.

- Не понял?!

- Я говорю, это паразиты, - незнакомец прикоснулся к лицу. - Личинки пауков. Я работал на Дикстре, где и пришелся по вкусу этим червякам. Здесь их пара миллионов. Период инкубации примерно десять лет - это и будет моим концом. Хотел бы я быть на торжественном обеде у своего недруга, когда меня разорвет, но так трудно все точно рассчитать! В общем, парень, я не виню тебя за то, что ты мне не доверяешь, но не надо судить только по внешности, Алекс. Ты ведь Алекс, не так ли? Я имею в виду, скажи ради бога, туда ли я попал?

- Да, я Алекс Райдер.

- А я Патрик Мак Греви. Я хочу сказать тебе только две вещи. Первое: убей змею. Приложи это привидение, которое преследует меня целых пять лет. Я не пилот. Кто я, не имеет значения, но таких, как я, больше, чем подсолнечных семечек, которые ты перевез за свою жизнь. Мы ждем возмездия и не можем получить его сами. Убей змею и ты поможешь всем нам.

Алекс не смог сдержать усмешки на своем лице, хоть и понимал, что менее всего сейчас пристало улыбаться. Он чувствовал, что им манипулируют. Им управляют, как запрограммированным роботом, заставляя ходить по замкнутому бессмысленному кругу. Что происходит вокруг него, черт побери?! Еще три месяца назад его счет побед фиксировался только на имитаторе. Краска на его лицензии еще не высохла и, тем не менее, какие то силы выбрали именно его орудием мести не опасному, а смертоносному противнику.

За ним наблюдает много людей и все они затаив дыхание чего-то ждут. Почему он?! Почему именно он?! И как сюда встраивается Элиссия?

- Хорошо, - спокойно сказал он. - Я понял. Но ты сказал: "Две вещи".

- Именно. Скажи, это Рейф посоветовал тебе провернуть операцию с шанаскилским мехом, как только ты сможешь ее осилить? Я прав?

Он был прав, это был один из последних ценных советов Рейфа, и Алекс его не забыл.

Мак Греви продолжал:

- Дав тебе такой совет, Рейф тем самым послал тебя ко мне. Тебе нужно подковать свою клячу. Ты должен взять на борт что-то действительно стоящее. Лети в Саут-Сити, найдешь центр частной торговли в Комплексе Магеллана.

- Я уже подковал свою клячу.

- Это ты так считаешь. Тем не менее, лети ко мне. Не упусти хороший шанс. Саут-Сити. Комплекс Магеллана…

Алекс колебался недолго. Он глянул на Эллссию. Пожав плечами, она кивнула, и Алекс принял предложение.

(Продолжение следует).

Содержание