Scan, OCR & spellcheck by
NUK, km, Василий Кормилицын,
Александр Кульнев, Lexys (Cybrex)
TXT version by Lexys (Cybrex)

ZX - РЕВЮ
N1,N2 1992

107241, Москва, Б-241, а/я 37, "ИНФОРКОМ"

Дорогие друзья!

Мы встречаем второй год существования "ZX-РЕВЮ". Позвольте поблагодарить наших верных читателей прошлого года, подписавшихся и на этот. Благодарим также и тех, кто в это трудное время поверил в нас, поверил в то, что мы будем жить и работать и впервые подписался на наше издание.

Мы, конечно, сохраним тот дух и стиль, которые были достигнуты в прошлом году, но кое-какие незначительные изменения все-таки внесем. Во-первых, уже в этом номере Вы увидите больше крупных учебных материалов, этим мы несколько парируем ту раздробленность, которая была в прошлом году, но в прошлом году мы готовили материал с "колес", а к этому году подошли с определенным портфелем и теперь можем давать более крупные вещи. В связи с этим будет как бы меньше число разделов в одном номере, но каждый из них будет крупнее. По всей видимости, это вызовет также необходимость полностью перейти на практику выпуска двойных номеров. Это, кстати, несколько поможет нам чуть-чуть снизить потери от резко возросших почтовых тарифов.

Поскольку в проекте намечены крупные материалы, мы будем их давать блоками, содержащими четное количество страниц. Это позволит Вам при желании без проблем расшивать выпуски и комплектовать свои подшивки по отдельным работам. Об этом просили многие читатели уже давно, но только сейчас появилась такая физическая возможность.

В этом году несколько больший упор будет сделан на основы программирования в машинных кодах и на Ассемблере.

Скорее всего, несколько уменьшится объем материалов технического характера, аппаратурных вопросов. Вы понимаете, что это связано, прежде всего, с тем, что мы не занимаемся аппаратным обеспечением и пользовались только тем, что нам давали партнеры.

А в общем "ZX-РЕВЮ" остаётся тем же, каким и было.

Мы по-прежнему, как и год назад, гарантируем всем читателям, подписавшимся на "РЕВЮ" возврат стоимости подписки, если кто-либо разочаруется в содержании этого номера и вернет нам его наложенным платежом, оценив в стоимость своих затрат и уведомив нас об этом письмом.

Многих волнует вопрос, как мы выйдем из того положения, в котором оказалась печать в масштабах всей страны, Вы ведь знаете, что вздорожавшая в прошлом году в 10 раз бумага сейчас поднялась еще в 5 раз. Невыносимыми стали почтовые тарифы. Достаточно сказать, что одна бандероль в Прибалтику теперь может стоить до 30 рублей. Люди волнуются, предлагают помощь. Многие пишут, что если нам будет совсем невмоготу, то они лучше доплатят, чем мы закроем наше дело. Большое спасибо, уважаемые читатели за вашу заботу, но пока мы держимся.

Наш принцип такой: "Когда дует сильный ветер перемен, то надо строить не забор, а мельницу".

Не успев начать подписной год, центральная пресса громко заголосила о невозможности выполнить взятые перед подписчиками обязательства и добилась дотаций от правительства, хотя голосить не перестала. Нам такой подход кажется не очень

красивым. Мы же работаем над другими проектами и сейчас для нас пока нет веских оснований предполагать, что нам придется просить Вас делать какие- либо доплаты, если мы нормально развернем маркетинг программного обеспечения для IBM. Хотя, конечно, в этом вопросе мы очень и очень рассчитываем на Вашу поддержку.

В №11-12 за прошлый год мы подробно расписали, как мы будем строить нашу с вами деятельность, предполагая начислять тем из Вас, кто помогает продвижению наших продуктов по 8% от объема продаж, проведённых по результатам Вашего исследования местного рынка.

Так же мы объявили о программном комплексе "АНГРАМ" (полный курс английского языка в 22-х программах) и предложили потенциальным дистрибуторам приобретать для представительства так называемый "ЗЕЛЕНЫЙ ПАКЕТ", в который входит образец программного средства, рекламно-информационные материалы, бланки договоров, инструкции и т.п.

Сейчас, когда пишутся эти строки, процесс уже пошел и по своим темпам превосходит наши самые смелые ожидания. Работа с дистрибуторами пошла. Если темп будет нарастать так, как сегодня, мы к осени этого года введем внутреннее специализированное издание, распространяющееся бесплатно "среди своих", содержащее анализ работы, обмен передовым опытом и описания наиболее сложных игровых программ для IBM-совместимых машин в качестве развлекательной части. Предполагаем, что записывать понравившиеся им программы наши дистрибуторы будут у нас бесплатно, а коллекция у нас - немалая. Одним словом, определяется наша маркетинговая политика на многие ближайшие годы.

Сегодня мы представляем на последней странице "ZX-РЕВЮ" не обучающую программу, а информационно-поисковую систему "РЕГИСТРАТУРА". Она выбрана для представления, поскольку обладает наиболее универсальным характером возможного применения. Наверное, очень трудно представить организацию, в которой был бы компьютер, чтобы она не нуждалась в подобной системе (разве что аналогичная уже есть). Система проходит опытную эксплуатацию уже более года в разных организациях в качестве системы учета кадров, системы ведения штатного расписания, системы учета клиентов в малом предприятии, системы учета обратившихся в медицинское учреждение, системы учета движения документов в делопроизводстве райисполкома, системы учета нуждающихся ветеранов войны и труда, системы учета учащихся на факультетах высшего учебного заведения. Мы внедрили ее даже на одном машиностроительном предприятии для учета заготовок. Это то, что уже есть на практике, а теоретически можно учитывать все и всюду.

Имеющийся опыт эксплуатации позволяет однозначно сказать, что среди систем с аналогичными возможностями, нашу выделяет необычайная простота в освоении и эксплуатации, она сделана для тех, кто впервые в своей жизни увидел компьютер. В этом мы видим свое главное достижение. Имея опыт преподавательской работы и методологию подготовки обучающих программ, мы и деловые системы делаем с расчетом на то, чтобы осваивать их можно было без головной боли и без необходимости изучать объемную сопроводительную документацию.

Далее, в последующих выпусках ZX-РЕВЮ мы будем чередовать представление обучающих и деловых систем. Ждем Вашего участия в дистрибуторской сети.

В заключение мы должны извиниться за возможную нерегулярность выпусков. Хоть мы и с оптимизмом смотрим в будущее, нам все-таки приходится, где возможно экономить и, к сожалению, мы вынуждены печатать тираж не тогда когда надо, а тогда, когда удается купить бумагу по более-менее сносным ценам, т.е. мы, конечно, не упускаем возможности уменьшить свои издержки. В то же время, нерегулярность не означает хроническое отставание, иногда мы, наверное, будем и забегать вперед.

Спектрум в школе

Сегодня раздел для начинающих представляют две БЕЙСИК-программы с комментариями, подготовленные Черкасским В. А. (г. Борисов).

Программы не сложны, но каждая из них содержит ошибки, попробуйте их отыскать.

1. "LISTNAME"

Эта программа записывается в начале кассеты и при работе выводит названия программ и их местоположение по счетчику.

```
10 DIM A$(50,3): DIM B$(50,10):DIM C$(50,4)
20 BRIGHT 1:  CLS: PRINT #0;INK 1: PAPER 6; AT 0,1;
      "BORISOV  $ ALCOSOFT $ 1991 ": PRINT AT 8,8;
      "1. Edit check"; AT  9,8;
      "2. Looking"; AT 10,8;
      "3. Save to the tape"
30 IF INKEY$="3" THEN GO TO 140
40 IF INKEY$="1" THEN GO TO 70
50 IF INKEY$="2" THEN GO TO 100
60 GO TO 30
70 CLS: PRINT " '0'-END"
80 INPUT "Number=";i:
      PRINT "       ";i;TAB(0):
      INPUT "Count="; A$(i)'
      "Name=";B$(i)'
      "DATA=";C$(i): GO SUB 150
90 GO TO 80
100 CLS: PRINT "Count      Name      Data": FOR  i=1 TO 50
110 BEEP .07,35: IF B$(i)="  " THEN PAUSE 0: GO TO 20
120 GO SUB 150
130 NEXT i
140 CLS: PRINT AT 10,10; "Save programs":SAVE  "LISTNAME" LINE 100:
      CLS: PRINT AT 10,10; "VERIFY programs": VERIFY  "LISTNAME":
      GO TO 20
150 PRINT A$(i);"_____"; B$(i); "_____";C$(i): RETURN
```

Пояснения к программе:

10 задаем строковые массивы для переменных:

A$ - показания счетчика;

B$ - названия программ;

C$ - год выпуска программы.

20 Выводим меню.

30-50 Выбор режима работы.

В зависимости от нажатой клавиши управление передается на разные строки программы.

60 Переход на строку 30 для организации ожидания нажатия клавиши.

70 Стирает экран, выводит условие выхода из режима.

80 Ожидается ввод порядкового номера, вывод его на экран и последовательно ввод показания счетчика, названия программы, года выпуска. Заканчивается обращением к подпрограмме вывода переменных на экран.

90 Организуется ввод каталога.

100 Инициализация экрана, вывод заглавия, начало цикла вывода данных.

110 Подача звукового сигнала, проверка на "пустое название", если "да", то пауза до нажатия любой клавиши и возврат в меню.

120 Обращение к подпрограмме вывода переменных.

130 Конец цикла FOR...NEXT. Пауза до нажатия клавиши и возврат в меню.

140 Очистка экрана, запись на ленту с последующим автостартом программы с 100 строки, очистка экрана, проверка записанной программы, возврат в меню.

150 Подпрограмма вывода переменных.

Ошибки:

Нужно добавить строку

```
145 IF i=0 THEN GO TO 100
```

и в строке 80 добавить после команды

```
INPUT "Number=";i : GO SUB 145
```

Это даст нормальный выход при программировании каталога и выведет название, которое пользователь будет редактировать. Кроме того, в 110 строке сравнение нужно проводить не с " ", а с " ", т.к. переменная B$(i) имеет фиксированную длину в 10 символов - это имя программы.

2. "REMONT"

Эта программа может служить пособием по ремонту различной техники. Причем, когда у машины варианты кончатся, то она спросит у пользователя, что неисправно, и в следующий раз она задаст и этот вопрос. Таким образом, постепенно количество вариантов увеличится.

```
5 GO TO 20
10 LOAD  ""CODE
20 DIM A$(500,32)
30 LET  i=1: LET A$(i)=" напряжение батареек"
40 CLS:  POKE 23607,249: PRINT "Я думаю, стоит проверить ";A$(i): GO SUB 140
50 IF UP=0 THEN PRINT FLASH  1';" Нужно ремонтировать!":PAUSE 0: GO TO  90
60 LET  i=i+1:IF A$(i)="  " THEN GO TO  80
70 GO TO 40
80 INPUT "Не хватает вариантов "' " как вы думаете, что еще нужно проверить ? " 'A$(i)
90 POKE 23607,60: CLS: PRINT AT 21,8; "Press any Key !": PRINT #0:
      INK 1; PAPER 6; AT 0,1;" BORISOV $ ALCOSOFT $ 1991":
      POKE 23607,249:PRINT AT 5,3;
      "1. - Работа с программой"; AT 5,3;
      "2. - Запись на магнитофон"; AT 7,3;
      "3. - Проверка записи"
100 IF INKEY$="1" THEN GO TO 30
110 IF INKEY$="2" THEN SAVE "REMONT" LINE 10
120 IF INKEY$="3" THEN VERIFY "REMONT"
130 GO TO 100
140 PRINT ," В норме?"'" Да/Нет?"
150 IF PEEK 23560=100 THEN LET UP=1:RETURN
160 IF PEEK 23560=110 THEN LET UP=0:RETURN
170 GO TO 150
```

Описание

5 Служит для того, чтобы после команды RUN программа нормально стартовала, а не производила загрузку по строке 10.

10 Загружает коды знакогенератора русского алфавита.

20 Объявляет строковый массив A$.

30 Объявляем переменную i и элемент массива A$(i).

40 Очищаем экран, переключаемся на русский шрифт, выводим вопрос и обращаемся к подпрограмме управления.

50 Если неисправность определена правильно, то выдаем сообщение.

60 Прибавляем 1 к i. Берем переменную A$(i) и проверяем ее на пустоту.

70 Организовываем переход для следующего вопроса.

80 Ждем вопроса, который вводит пользователь.

90 Вывод меню.

100 Опрос клавиши "1". Если нажата, то работа с программой.
110 То же клавиши "2". Если нажата - выгрузка программы на ленту.
120 То же клавиши "3". Если нажата - проверка программы.
130 Организовывает ожидание INKEY$.
140 Начало подпрограммы определения ответа.
150 Опрашивается клавиша "D".
160 То же - клавиша "H".
170 Организация ожидания INKEY$. Знакогенератор находится в памяти по адресам 64000... 64768.

Ошибки.

1. В строку 150 надо дописать POKE 23560,0 для того ,чтобы программа не проскакивала INKEY$.
2. В строке 60 сравнение должно быть с пустой строкой длиной 32 символа.

BETA BASIC

БЕТА-БЕЙСИК 3.0 - дальнейшее развитие серии, начатое программами БЕТА - БЕЙСИК 1.0 и 1.8. Как и последние, он полностью совместим со стандартным, зашитым в ПЗУ вашего компьютера БЕЙСИКОМ, но обладает значительно большими возможностями, определенной гибкостью и имеет значительное число новых операторов и функций.

БЕТА-БЕЙСИК откроет перед вами гигантские перспективы. Во-первых, с его помощью Вы можете писать большие программы. Если, Вы не пробовали на обычном БЕЙСИКе писать программы более 10 - 15 K, то знайте, что "проклятьем" обычного БЕЙСИКа является отсутствие структурного программирования, наступает момент, когда программа не укладывается, как цельное произведение в вашем мозгу и тогда любые изменения, производимые в одном месте программы, влекут за собой помехи, появляющиеся в другом месте, как бы ни был прост БЕЙСИК в освоении, но в отладке большие программы, написанные на БЕЙСИКе, могут выходить за все разумные рамки по трудоемкости.

Второе преимущество БЕТА-БЕЙСИКА - отличные средства редактирования. Этот момент, надо сказать, ахиллесова пята стандартного БЕЙСИКа "Спектрума". Конечно, строчный редактор со сложной системой вызова строки на обработку - это не дело. Только на этом программист теряет половину своего драгоценного времени. В то же время, полноэкранные редакторы, какие мы виден в БЕЙСИКе на хорошо продвинутых машинах типа MSX или, скажем, IBM (GW_BASIC) - это тоже не подарок. Перемешанные на одном экране редактируемые и нередактируемые строки приводят к многочисленным ошибкам, да и занимают такие редакторы много места.

Тот подход, который развит в БЕТА-БЕЙСИК 3.0 наверное наиболее практичный и эргономичный.

Третье преимущество БЕТА-БЕЙСИКА - в его экономном расходовании памяти. Да, конечно, сам он занимает до 20K. То есть почти половину того, что имеет Ваш компьютер. Но, если Вы, программируя, будете хорошо использовать его возможности, то затратите 4-5 K на основные процедуры, а далее, используя их, сможете в каждый килобайт укладывать то, что на обычном БЕЙСИКе стоило бы вам пяти-семи K. Таким образом, в доступных вам 20 килобайтах, Вы сможете разместить примерно 10K данных плюс 4-5K процедур, плюс программу, эквивалентную обычным 40K стандартного БЕЙСИКа. Конечно, это цифры условные, но условность их в том, что это далеко не предел. Поработав несколько недель, Вы сможете еще сильнее повысить емкость программ, это просто данные нашего собственного практического опыта в "Инфоркоме".

Данный документ является нашим авторизованным переводом фирменной инструкции. Авторизованность не означает, что мы что-то сократили. Мы чуть более подробно прокомментировали наиболее узкие места и заменили (там, где это можно) те примеры, которые были привязаны к микродрайву на более близкие для нашего читателя.

Пусть Вас не смущает объем документа, нет никакой необходимости читать все от начала и до конца, вполне достаточно, если Вы прочтете введение, раздел по редактированию программ и, в дальнейшем, будете обращаться к документу только при необходимости использовать тот или иной оператор или ту или иную, ранее неизвестную Вам функцию. Особо же рекомендуем вам обратить внимание на те преимущества, которые предоставляет структурное программирование благодаря использованию процедур.

ГЛАВА 1. ВВЕДЕНИЕ

Загрузка языка с ленты выполняется обычным образом:

```
LOAD "" или LOAD "Beta Basic"
```

По этой команде загружаются три первых БЕЙСИК-строки - строка 0, строка 1 и строка 2. Со строки 2 осуществляется автостарт программы, в результате которого загружается остальная (основная) часть БЕТА-БЕЙСИКа, представляющая из себя блок машинных кодов длиной 18K. Системная переменная RAMTOP понижается до значения

47070, предохраняя от повреждения машинный код, размещаемый в верхней части памяти.

После загрузки программы на экране должно появиться исходное системное сообщение. При этом строки 1 и 2 программы удаляются и остается только нулевая строка.

Внимание! Если Вы написали программу под управлением БЕТА-БЕЙСИКа и желаете передать ее своим знакомым для эксплуатации, то пожалуйста перед выгрузкой Вашей программы и машиннокодового блока внесите следующие изменения в машиннокодовую часть БЕТА-БЕЙСИКа:

```
POKE 64218,0: POKE 64219,0: POKE 64220,0: POKE 64861,201
```

Этим Вы обеспечите то, что Ваша программа будет запускаться, но не сможет быть изменена.

Особенности работы с языком БЕТА-БЕЙСИК 3.0

В строке 0 содержатся новые определения функций БЕТА-БЕЙСИКа 3.0. Эта строка всегда присутствует в программе, хотя и не всегда изображается на экране. Характерным свидетельством того, что БЕТА-БЕЙСИК загружен и готов к работе является изображение указателя текущей строки в инвертированном виде. Несколько увеличена и продолжительность звукового сигнала, подаваемого при нажатии клавиш. Если он вам не нужен, дайте команду POKE 23609,0 для отключения. Теперь можете загружать любую БЕЙСИК-программу (доступного размера) и она будет работать нормально, за исключением двух особенностей:

1. Загрузку программы надо выполнять не командой LOAD, а командой MERGE, иначе погибнет нулевая строка. Конечно, если программа была написана под стандартным БЕЙСИКОМ, ей это все равно, но если под БЕТА-БЕЙСИКОМ, строку 0 уничтожать не надо.

2. Если вам надо, чтобы вместо ключевых слов БЕТА-БЕЙСИКа на экране изображались символы блочной графики, надо дать команду KEYWORDS 0 (см. соответствующую команду).

Еще одной особенностью является то, что несколько изменилось изображение программных строк, получаемое на экране по команде LIST, если длина строки больше ширины экрана. В этом случае в первых четырех позициях печатаются только номера строк.

Несколько уменьшено разрушительное действие команды NEW. Она удаляет из памяти имеющуюся там БЕЙСИК-программу, но оставляет строку 0. Если же Вы хотите удалить из памяти и сам БЕТА-БЕЙСИК, то остается только сделать RESET, если у Вас есть такая кнопка, или выключить питание или переинициализировать машину командой RANDOMIZE USR 0.

Несколько увеличена скорость выполнения программ по сравнению со стандартным БЕЙСИКОМ, особенно длинных.

Циклы FOR-NEXT работают с одной и той же скоростью, независимо от местоположения в программе, что является отличием от стандартного БЕЙСИКа. Наивысшая скорость достигается, когда начало цикла, его длина и конец выражены целыми числами (в том числе и отрицательными) размером до 65535. Такие циклы работают в 5 раз быстрее, по сравнению с циклами, размещенными до 100-й строки в стандартном БЕЙСИКе и в 17 раз быстрее, по сравнению с размещенными до 500-ой строки в стандартном БЕЙСИКе.

Быстрее работают и GO TO и GO SUB, особенно когда строка назначения отстоит достаточно далеко. Быстрее работает и RETURN. Возврат к последней строке выполняется столь же быстро, как и к первой, в отличие от стандартного БЕЙСИКа.

В какой-то степени скорость удалось повысить за счет того, что интерпретатор запоминает адреса, а не номера строк. С другой стороны, это накладывает определенные ограничения на редактирование. Так, если Вы, например, остановите программу во время исполнения цикла, введете внутрь этого цикла дополнительную строку, а потом запустите программу дальше - CONTINUE, то CONTINUE может не сработать, поскольку адрес возврата изменится, хотя номер строки возврата не изменялся.

Работа с более ранними версиями БЕТА - БЕЙСИКа

Если Вы загрузите программу, написанную в более ранней версии БЕТА-БЕЙСИК 1.0

или БЕТА-БЕЙСИК 1.8, то вместе с ней загрузится и нулевая строка, содержащая определения функций языка. Если теперь Вы хотите работать с этой программой под управлением версии 3.0, то Вам надо заменить старую версию нулевой строки на новую.

1. Загрузить БЕТА-БЕЙСИК 3.0.
2. Загрузить программу, выполненную в версии 1.0 или 1.8.
3. Выполнить "MERGE" для БЕТА-БЕЙСИКа 3.0, чтобы ввести копию строки 0.
4. Удалить строки 1 и 2.
5. Выгрузить полученную программу.

Есть два момента несовместимости версии 3.0 с ранними версиями. Во-первых, переменная под именем "line", создававшаяся операторами ON ERROR и TRACE, теперь называется "lino", чтобы исключить путаницу с ключевым словом стандартного БЕЙСИКА LINE.

Во-вторых, в новой версии имена процедур не могут иметь внутренних пробелов.

Преобразовать Вашу программу из версии, написанной на более ранней версии языка, Вы можете воспользовавшись командой ALTER.

Для тех, кто знает версии 1.0 и 1.8, желательно перечитать работу с командами ON, RETURN, ROLL, SCROLL, CLOCK, ON ERROR и TRACE, поскольку они изменены. Несколько изменена и команда GET (число). Значительно расширены возможности процедур.

Загрузка и выгрузка программ, написанных в БЕТА-БЕЙСИКе 3.0.

После того, как Вы написали программу, в которую входят некоторые новые ключевые слова, Вы можете выгрузить ее на ленту обычным порядком. Если теперь вам когда-либо понадобится вновь загрузить ее, сначала загрузите БЕТА-БЕЙСИК 3.0, если он еще не загружен. Теперь загрузите (LOAD) свою программу. Поскольку строка 0 была выгружена вместе с программой, то теперь нет необходимости использовать MERGE.

С другой стороны, Вы можете воспользоваться строками 1 и 2 загрузчика для того, чтобы сохранить на ленте свою программу вместе с машинно-кодовой частью БЕТА - БЕЙСИКа. В этом случае Ваша программа и БЕТА-БЕЙСИК будут загружаться автоматически.

Если Вы загрузите программу, написанную на БЕТА-БЕЙСИКе, в компьютер в то время, как в нем не присутствует машинный код БЕТА-БЕЙСИКа, новые команды появятся в виде одиночных символов, а после команды RUN Вы получите сообщение об ошибке "Nonsense in Basic". В этом случае Вам следует использовать: MERGE "Beta Basic": GO TO 2, чтобы загрузить загрузчик БЕТА БЕЙСИКА, а из него по автостарту посредством GO TO 2 загрузится и машинно-кодовая часть.

ГЛАВА 2. РЕДАКТИРОВАНИЕ

Список используемых ключевых слов: EDIT, KEYWORDS, LIST, FORMAT, CSIZE, JOIN, SPLIT.

БЕТА-БЕЙСИК позволяет делать ввод и редактирование программ намного более простым, чем то, к чему Вы привыкли, работая со стандартным встроенным БЕЙСИКОМ "Спектрума". Если Вы наберете или прильете (MERGE) некую БЕЙСИК программу, то сможете поэкспериментировать с новыми возможностями. Поскольку все возможности стандартного БЕЙСИКа сохранены. Вы вряд ли будете испытывать при этом какие-либо трудности. Ниже мы рассмотрим, как действуют те команды, которые добавляет использование третьей версии БЕТА-БЕЙСИКа при редактировании. Более подробно мы на них остановимся еще раз в основной части руководства.

Курсор текущей строки

Первое, что Вы заметили, загрузив БЕТА-БЕЙСИК, - это курсор текущей строки, изображающийся инверсным цветом на экране. Его можно перемещать с помощью курсорных клавиш вверх и вниз и выполняется это намного быстрее, чем в стандартном БЕЙСИКе, поскольку при этом не листается содержимое всего экрана. (Могут быть редкие случаи, когда положение курсора не вполне соответствует изображению текста программы

на экране. В этом случае просто нажмите ENTER для перестроения экрана).

Команда EDIT <номер строки>

Вы можете вызвать на редактирование даже ту строку, которой в данный момент нет на экране. Теперь нет необходимости подгонять к ней курсор. Достаточно нажать "0" и набрать нужный номер строки, чтобы она появилась в нижней части экрана.

В редактируемой строке Вы теперь можете перемещать курсор не только влево или вправо, но и вверх и вниз, это опять же выполняется курсорными клавишами. При попытке поднять курсор выше верхней строки или опустить ниже нижней, он автоматически встанет в конце строки. Самый быстрый путь добавить операторы в конец Вашей строки - это:

- нажать клавишу "0";
- ввести номер строки;
- нажать "Курсор вверх" - теперь он окажется в конце строки.

Переключение режимов курсора

Теперь Вы можете в нужный момент легко переключить курсор "K" на курсор "L" или "C". Это бывает полезным в тех случаях, когда Вы хотите набрать по буквам имя процедуры или если Вы не желаете вводить ключевые слова как токены, то есть одним нажатием клавиши, а хотите набирать их по буквам, что возможно благодаря команде KEYWORD (см. ниже). Выполняется переход из режима "K" в режим "L/C" нажатием клавиши "пробел".

Возможен и обратный переход из режима "L/C" в режим "K", что выполняется одновременным нажатием клавиш SYMBOL SHIFT и ENTER. Те, кто внимательно читают "ZX-РЕВЮ", знают, что прямым путем в стандартном БЕЙСИКе это невозможно. Для этого мы набирали оператор THEN, а потом стирали его (см. "Маленькие хитрости" ZX РЕВЮ-91, стр.52). Это бывает полезно, если Вы работает в режиме: с отключенными токенами (KEYWORDS 4 см. ниже) или если Вы хотите ввести ключевое слово в строковую переменную, что бывает полезным при работе с командами REF, ALTER или KEYIN.

Управление вводом ключевых слов

Команда KEYWORDS позволяет Вам переключать режим ввода ключевых слов, т.е. вводить их одним нажатием клавиши, как в стандартном БЕЙСИКе, или набирать полностью по буквам, как это делается на компьютерах иных систем. Есть и режим KEYWORDS 3 в котором в одной строке можно одновременно применять и тот и другой подход. Это тоже может быть полезным. Даже если Вы хорошо знаете стандартную систему набора и Вам нравится набирать слова типа RANDOMIZE одним нажатием клавиши, все же ввести оператор IN по буквам несколько проще. Причем набор может идти как прописными, так и строчными буквами.

Ввод ключевых слов БЕТА-БЕЙСИКа

Есть два способа ввода ключевых слов БЕТА-БЕЙСИКа. Вы можете набирать их по буквам, используя ведущий пробел для того, чтобы отключить курсор "K", если необходимо, или можете использовать "одноклавишный" подход. В последнем случае команды и функции вводятся по-разному. Для ввода новой команды сначала перейдите в графический режим, а затем нажимайте соответствующую клавишу. Большинство клавиш в этом случае дают новые ключевые слова.

Для ввода новой функции наберите FN, а затем - "$" или "(" - в зависимости от того, что это за функция). Набирать "FN" можно теперь по-разному. Во-первых обычным порядком, как ключевое слово стандартного БЕЙСИКа, во вторых по буквам "f" + "n" + " " и, в-третьих, - нажав клавишу "Y" в графическом режиме.

Проверка синтаксиса

БЕТА-БЕЙСИК, как и стандартный БЕЙСИК, проверяет правильность того, что Вы вводите в компьютер и точно так же выдает звуковой сигнал BEEP, если устанавливает наличие ошибки. Изменить звуковой сигнал Вы можете, изменяя значения в ячейке памяти 23608 посредством POKE 23608,.. . Звуковой сигнал удобен, если Вы набираете программу,

например, из распечатки в журнале и при этом не часто смотрите на экран. Но имейте в виду, что если Вы работаете в режиме набора ключевых слов по буквам, то ошибки в их написании будут восприняты компьютером как ввод нового имени процедуры. Например, если вместо PAPER 1 Вы наберете PAPRE 1, то получите сообщение об ошибке "Missing DEF PROC" (отсутствует определение процедуры).

Команда LIST FORMAT

Эта команда позволяет Вам улучшить читабельность распечатки Вашей программы на экране монитора.

Команда CSIZE

Позволяет Вам уменьшить размер символов, доведя их количество в строке до 64 или, наоборот, увеличить их размер для печати заголовков или в иных, например в демонстрационных целях.

Команды JOIN <номера строк> и SPLIT.

Позволяют объединить две строки в одну или, наоборот, разбить строку на две.

Управляющий код "новая строка".

Вы можете ввести этот управляющий код в свою программную строку или в строку INPUT путем нажатия CAPS SHIFT и ENTER. В отличие от просто ENTER Вы сможете продолжать ввод той же программной строки на другой экранной строке. Смотрите также раздел "Управляющие коды" (CHR$ 15).

ГЛАВА 3. ПРОЦЕДУРЫ

Список используемых ключевых слов: DEF PROC, END PROC, LOCAL, DEFAULT, REF, READ LINE, LIST PROC и функция ITEM().

В этой главе мы кратко рассмотрим вопросы, связанные с применением процедур в БЕТА-БЕЙСИКе. Более подробную информацию Вы сможете получить далее, прочитав разделы относящиеся к каждому из указанных ключевых слов.

Использование процедур - весьма желанный прием для тех, кто программирует на БЕЙСИКе. Современные версии БЕЙСИКа в той или иной мере используют эту возможность и программисты встречают их с большим энтузиазмом. И, если Вы ими не пользуетесь, то наверное стоит пересмотреть свои взгляды, причем вовсе не потому, что это модно или потому, что это все рекомендуют, а просто так удобнее и легче писать и отлаживать программы.

БЕТА-БЕЙСИК 3.0 имеет одну из самых совершенных систем использования процедур среди других языков для домашних персональных компьютеров. Именно здесь Вы получаете наибольшую гибкость и эффективность программирования.

Основное преимущество использования процедур состоит в структурном программировании. Эта концепция предполагает, что Вы можете создать некоторый программный модуль, способный выполнять определенную работу и не оказывать никаких нежелательных побочных эффектов на остальную часть программы. Он не должен, например, изменять никакие переменные в программе, кроме тех, которые положено. После того, как этот блок программы вами написан и тщательно отлажен, Вы можете забыть о том, как он работает и из чего состоит. Когда Вы пишете новую программу и он Вам нужен, Вы просто пришьете его к тексту командой MERGE "", а используете - вызвав его по имени. Каждая процедура должна быть достаточно простой, чтобы быть вполне понятной. Если она у Вас получается громоздкой, сложной и непонятной, то стоит подумать о том, чтобы разделить ее на несколько логических частей и выразить каждую из этих частей своей процедурой. Процедура - это часть программы, имеющая свое имя и начинающаяся с оператора DEF PROC, после которого задается ее имя и имена тех переменных, с которыми она должна работать, а заканчивающаяся оператором END PROC.

Давайте рассмотрим простой, хотя и вполне бесполезный пример:

```
100 DEF PROC greet
```

```
110 PRINT "Hello"
120 END PROC
```

Попробуйте дать команду RUN и Вы увидите, что ничего не произойдет. Хоть процедура и часть программы, но работать в таком виде она не будет. Здесь мы записали только определение процедуры, то есть указали, что она должна делать. Выполнение процедуры происходит только после вызова ее по имени. Наша процедура имеет имя "greet". Попробуйте добавить строку

```
10 greet
```

Теперь, если Вы дадите команду RUN, в строке 10 будет запущена эта процедура и, в соответствии с ее определением, будет напечатано на экране слово Hello.

Все происходит точно так же, как в стандартном БЕЙСИКЕ с функциями. Там ведь тоже DEF FN игнорируется до тех пор, пока программа не встретит вызов функции оператором FN.

Вы можете столкнуться с проблемой того, как набрать слово greet в строке 10, ведь курсор находится в режиме "K", выше мы об этом упоминали:

- Вы можете нажать пробел и далее набирать greet по буквам;

- Вы можете дать команду KEYWORDS 4 и перейти в режим ввода всех ключевых слов по буквам;

- Вы можете использовать ключевое слово PROC и записать строку в виде:

```
10 PROC greet
```

Это совершенно то же самое, что и просто

```
10 greet
```

Использование ключевого слова PROC в данном случае просто дань привычке тех программистов, которые привыкли к такой записи по более ранним версиям БЕТА-БЕЙСИКА, хотя реально в нем больше нет никакой необходимости.

Далее мы будем говорить об использовании имени процедуры, как о вызове процедуры. Имя процедуры обязательно должно начинаться с буквы. Заканчивается имя процедуры пробелом, двоеточием, нажатием ENTER, операторами REF или DATA. Остальные символы, как правило, могут быть использованы в имени процедуры, но желательно, чтобы Вы привыкли использовать только буквы, цифры и символы подчеркивания - "_". Не имеет значения, какие буквы применяются строчные или прописные.

Определение процедуры начинается с DEF PROC и может быть расположено где угодно в программе. Его вполне можно располагать и до и после первого вызова процедуры. Важно только, чтобы оператор DEF PROC был первым оператором в строке. Определение процедуры может иметь сколько угодно строк и должно заканчиваться оператором END PROC.

Если Вы используете с одним DEF PROC несколько END PROC, то компьютер будет не в состоянии правильно "перепрыгнуть" через определение процедуры во время работы программы. В этом случае Вам придется самостоятельно позаботиться, чтобы он смог это сделать или размещать все лишние END PROC после оператора STOP. Изменим приведенный выше пример:

```
100 DEF PROC greet times
110   FOR n=1 TO times
120     PRINT "HELLO"
130   NEXT n
140 END FROC
```

В данном случае times - это параметр процедуры, он называется формальным параметром, поскольку не имеет пока никакого числового значения, а только указывает, как он используется при расчете процедуры. Раз он указан в определении процедуры, то он должен быть задан и при вызове процедуры. Теперь, когда вам надо будет исполнить процедуру greet, Вы зададите параметр times и заданное вами значение называется фактическим параметром - это число, которое фактически будет подставлено на место формального times при расчете процедуры.

Итак, формальные параметры - это просто имена, а фактические параметры - это числа, выражения или переменные (кроме тех случаев, когда используется REF, о чем см.

ниже).

Теперь, если Вы обратитесь к процедуре по имени, без задания параметра, например greet, то получите сообщение об ошибке "Variable not found" (переменная не найдена).

Если же Вы обратитесь к процедуре:

```
10 greet 5
```

Вы увидите что слово Hello будет пять раз напечатано на экране, т.к. фактический параметр "5" встал на место формального times.

Мы надеемся, что тем, для кого применение процедур в программах является новым делом, пока все понятно, но надо обсудить еще один очень важный момент, который может в больших программах повлечь за собой трудно обнаруживаемые ошибки.

Дело в том, что, как мы говорили, процедуры должны исполнять то, что записано в их определении и не должны оказывать нежелательных побочных воздействий на остальную часть программы. Рассмотрим нашу процедуру. Она манипулирует с двумя переменными times и n. Давайте теперь проверим, чему они равны после того, как процедура отработала. Дайте прямые команды:

```
PRINT times
PRINT n
```

Вы обнаружите, что times - не существует, а n - существует и имеет числовое значение.

Почему не существует times? Дело в том, что эта переменная введена нами в строке DEF PROC, что автоматически сделало ее локальной, то есть определенной только внутри процедуры. Когда процедура кончила работать, она удаляется из памяти и ее больше нет. И это хорошо, ведь если у Вас где-то еще в программе используется какая-нибудь переменная times, то ее значение могло бы быть изменено (испорчено), а теперь вам можно об этом не думать. А если у Вас до вызова процедуры уже была какая-то переменная times, то она тоже будет удалена? Нет, в момент вызова она будет запомнена, как глобальная переменная, а в процедуре будет создана локальная times. После работы процедуры локальная будет удалена, а глобальная - восстановлена. Хуже обстоит дело с переменной n, - она осталась существовать после работы процедуры и является глобальной, ведь она не входила в оператор DEF PROC. Ее значение может незаметно для Вас быть использовано где-то еще и привести к неприятным ошибкам. Для того, чтобы это не происходило, ее можно объявить локальной переменной внутри процедуры в принудительном порядке. Для этого служит оператор LOCAL.

Добавьте к нашей программе строку:

```
105 LOCAL n ,
```

а теперь добавьте к программе следующие строки, которые позволят вам убедиться, что после работы процедуры переменные times и n остались ненарушены:

```
10 LET n=1234
20 LET times=5678
30 greet 10
40 PRINT n,times
```

Процедура "greet" имела скорее учебное, чем практическое назначение, а вот более полезная процедура:

```
100 DEF PROC box x,y,width,height
130 PLOT  x,y: DRAW width,0
140 DRAW 0,-height: DRAW -width,0
150 DRAW 0,height
160 END PROC
```

Процедура "box" предназначена для рисования прямоугольника и имеет четыре параметра: x, y - координаты левого верхнего угла, width - желаемая ширина, а height - высота. Так, команда box 100, 100, 10, 40 изобразит вблизи центра экрану прямоугольник, вытянутый по вертикали.

Теперь предположим, что мы хотим, чтобы у нас был квадрат, а поскольку у него высота равна ширине, то попробуем ее не указывать, например:

```
box 100, 100, 50
```

Получим сообщение об ошибке, т.к. процедура имеет четыре формальных параметра, а мы им на смену подставили только три фактических. Это, однако, тоже можно

предусмотреть и предотвратить, для чего служит оператор default (по английски default - принятый "по умолчанию"). Запишем строку:

```
120 DEFAULT height=width
```

Эта строка буквально означает следующее: "Если параметр height не задан, то считать, что он равен параметру width".

Ну, а если Вы уж совсем ленивы, то можете не указывать и ширину квадрата, введя строку:

```
110 DEFAULT width=20
```

и у Вас и ширина и высота будут равны 20 пикселам. Используя запятые, можно опускать не только последние параметры, но и вообще любые.

```
box ,100,40,10
```

В этом вызове опущен параметр x. Разумеется, где-то в описании процедуры должен присутствовать оператор

```
DEFAULT x = ...
```

Передача параметров в виде ссылки

Итак, мы рассмотрели, как информация передается из главной части программы в процедуры с помощью параметров. Фактические параметры заменяют формальные, определенные в операторе DEF PROC. Но у нас может возникнуть необходимость не только передавать что-то в процедуру, но и, например, получать что-то от нее. Можно, конечно, написать процедуру, которая будет что-то рассчитывать, а результат расчета присваивать глобальной переменной x, из которой можно этот результат узнать после окончания работы процедуры и возврата в главную программу. Но это нарушит нашу договоренность о том, что процедура должна быть независимым модулем и не должна оказывать нежелательного влияния на другие участки программы (и на другие процедуры тоже). А если мы поступим таким образом с переменной x, то теперь должны будем все время помнить о том, что эту букву уже нигде нельзя использовать для иных целей, т.к. можно потерять то, что в этой переменной содержится. Желательно было бы уже при вызове процедуры и указании фактических параметров задать какую-то переменную, в которую надо поместить результат работы процедуры. Данная версия БЕЙСИКа, в отличие от многих аналогов, позволяет делать и это.

Обычно, при вызове процедуры в нее передаются параметры в виде значений (чисел), которые встают на место формальных параметров, но можно и передать просто имя переменной, без указания ее значения. Это называется передачей параметра, как ссылки, и делается добавлением оператора REF перед именем этой переменной в операторе DEF PROC. REF - это сокращение от английского слова reference (ссылка). Таким образом, переменная, которую Вы ставите при вызове процедуры в качестве фактического параметра, переименовывается в то имя, перед которым стоит REF, вычисляется в процедуре и возвращается под первоначальным именем в вызывающую программу или процедуру. В качестве демонстрации приведем процедуру SWOP, которая обменивает между собой содержимое двух строковых переменных.

```
200 DEF PROC SWOP REF a$, REF b$
210 LOCAL  t$
220 LET t$=a$: LET a$=b$: LET b$=t$
230 END PROC
```

А вызывается она, например, так:

```
10 LET x$="hi":LET y$="goodbye"
20 SWOP x$,y$
30 PRINT x$,y$
```

Без указания REF в определении функции и a$ и b$ тоже будут обмениваться своим содержимым в процедуре SWOP, но только именно внутри нее: глобального эффекта не будет, т.к. локальные a$ и b$ при выходе из процедуры будут утрачены. С использованием же REF эти переменные являются как бы временными именами, на которые ссылаются глобальные x$ и y$, поэтому изменения которым подверглись a$ и b$ в теле процедуры, отражаются и на x$ и на y$.

Языки программирования для микрокомпьютеров, использующие концепции

процедур, в большинстве случаев не позволяют использовать массивы в качестве параметров. БЕТА-БЕЙСИК разрешает это, но правда, требует, чтобы они передавались только как ссылки. Они переименовываются вместо того, чтобы просто копироваться в область локальных переменных процедуры. Этим достигается экономия памяти, ведь компьютеру не надо одновременно хранить один массив два раза. Если же вам на самом деле нужна локальная копия Вашего массива для каких-то временных манипуляций с ним, Вы можете внутри процедуры создать параллельный массив, объявив его как LOCAL, а затем скопировать в него содержание Вашего исходного массива, воспользовавшись для этого командой БЕТА-БЕЙСИКа COPY, о чем мы еще скажем ниже, когда будем ее рассматривать вместе с командой JOIN. Вот демонстрационный пример, в котором показано, как можно найти сумму элементов массива.

```
300 DEF PROC total REF a(),REF sum
310 LOCAL n
320 LET sum=0
330 FOR n=1 TO LENGTH  (1,"a()")
340 LET sum = sum + a()
350 NEXT n
360 END PROC
```

За именем массива должны идти скобки, чтобы интерпретатор отличал имена массивов от обычных переменных с тем же именем. Перед "sum" стоит оператор REF, так что по окончании работы содержимое "sum" будет передано глобальной переменной. Функция LENGTH (), о которой мы еще будем говорить, определяет размер массива для того, чтобы процедура могла работать с массивами любой длины.

Теперь зададим сам массив, чтобы убедиться, что наша процедура работает нормально:

```
100 DIM t(10)
110 FOR n=1 TO 10
120 LET t(n)=n
130 NEXT n
```

и добавим вызов нашей процедуры:

```
140 total t(), answer
150 PRINT answer
```

в итоге получим 55.

Передача параметров списком

Возможны варианты, когда вам вместо того, чтобы определять комбинацию параметров для процедуры, удобнее иметь дело со списком этих параметров, причем список может быть неопределенной длины. Чтобы это было возможным, в БЕТА-БЕЙСИКе 3.0 есть специальные средства.

Если в операторе DEF PROC использовать оператор DATA вместо обычного перечисления формальных параметров, то соответствующий ему оператор READ примет список фактических параметров, стоящих в вызове вашей процедуры. Чтобы эта возможность была по настоящему удобной, необходимо, чтобы можно было определить в процедуре есть ли еще параметры в списке, которые она не приняла. Для этого существует функция ITEM(). Она возвращает 0, если список исчерпан полностью, 1 если в списке есть не переданные параметры и следующий параметр - число, 2 - если следующий параметр - строковая переменная.

Приведен пример, который показывает, как можно организовать процедуру, которая просуммирует несколько заданных вами чисел.

```
100 DEF PROC SUM DATA
110 S = 0
120 DO UNTIL ITEM()=0
130 READ a
140 s = s+a
150 LOOP
160 PRINT sum
170 END PROC
```

Примененные в строках 120, 150 операторы DO UNTIL и LOOP - это удобная форма

организации цикла (см. далее). В принципе вместо DO UNTIL можно было бы применить и DO WHILE (см. ниже).

Вызов этой процедуры можно выполнить, например так:

sum 1, 2, 3, 4 или с других количеством параметров:

sum 1, 2, x, y, z, 1256

Если Ваш список параметров состоит из строковых переменных, то в строке 130 надо было бы применять READ a$, а не READ a. А как быть, если список смешанный и содержит и числа и строки? В этом случае перед READ надо проверить, очередной параметр с помощью ITEM() и использовать либо тот вариант, либо другой. В строковых переменных при этом можно избежать использования кавычек, если вместо READ использовать READ LINE (см. далее).

Обратите внимание на то, что список фактических параметров, передаваемых через DATA и READ, исключает возможность им быть локальными, если Вы специально это не зададите.

Рекурсия

Процедура, которая вызывает саму себя, называется рекурсивной. Есть анекдот, что в одном из компьютерных словарей-справочников написали:

РЕКУРСИЯ см. РЕКУРСИЯ.

Рекурсия часто помогает очень элегантно избегать сложных программистских проблем. С другой стороны, это не самая быстро работающая структура. К тому же возможны большие расходы памяти. Ведь при каждом вызове процедурой самой себя формируется новый временный список локальных переменных и для них выделяется память. Кроме того, несмотря на внешнюю простоту такой структуры, ее тщательный разбор может вызвать у программиста легкое головокружение. В качестве примера мы приведем процедуру, которая рисует бриллиант, а затем еще один такой же, но поменьше. Внутри первого и еще один внутри второго и так далее, на минимальный размер наложено ограничение, иначе процесс мог бы продолжаться бесконечно.

```
100 DEF PROC diamond x,y,size,diff
      DEFAULT diff=15
      PLOT x,y,-size
      DRAW -size,size
      DRAW size,size
      DRAW size,-size
      DRAW -size,-size
110 IF size >4 THEN
      diamond x,y+size,size-diff
      diamond x,y-size,size-diff
      diamond x-size,y,size-diff
      diamond x+size,y,size-diff
130 END PROC
```

Вызвать эту процедуру можно, например, так:

```
diamond 128,88,40
```

Ошибки

Если Вы попробуете вызвать процедуру, которую забыли задать, Вам дадут сообщение об ошибке W: "Missing DEF PROC". Если же Вы забудете закрыть процедуру с помощью END PROC - сообщение X: "No END PROC". Программа будет стараться во время работы "перепрыгнуть" через блок, в котором задается процедура и не сможет этого сделать.

Если при вызове задано больше фактических параметров, чем их есть в наличии в описании процедуры, - сообщение "Parameter error." Если тип формальных параметров не совпадает с типом реально установленных фактических параметров - "Nonsense in BASIC". Оператор END PROC может генерировать сообщение "Variable not found", если оказывается, что переменная, которую процедура должна передать в качестве выходного параметра, не существует.

ГЛАВА 4. СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

Кроме заложенной в БЕТА-БЕЙСИК 3.0 концепции использования процедур, о чем мы только что написали, введены еще дополнительно некоторые средства, обеспечивающие возможность структурного программирования: операторы DO,LOOP, EXIT IF, WHILE и UNTIL обеспечивают те же дополнительные возможности организации циклов, что и REPEAT и WHILE в более ранних версиях еще больше повышают гибкость в программировании.

В структуре операторов IF…THEN можно использовать ELSE.

Оператор ON позволяет выбрать необходимый номер строки. Это как бы упрощенная форма операторов CASE или SWITCH.

Команда LIST FORMAT может вам обеспечить вывод текста программы в структурированной форме ("лесенкой") так, как это принято в языках, поддерживающих структурное программирование (ПАСКАЛЬ, СИ и др.).

ГЛАВА 5. ОБРАБОТКА ДАННЫХ

В данной версии языка введены дополнительно следующие операторы обеспечивающие обработку данных разного типа:

JOIN, COPY, DELETE, SORT - для работы с массивами и со строковыми переменными.

INARRAY и INSTRING - поисковые функции.

Функции LENGTH, CHAR, NUMBER.

Операторы: EDIT <переменная>, SAVE DATA, USING.

Функции: EOF, SHIFT$, USING$.

JOIN и COPY позволят вам перемещать или копировать массив данных или его часть в другой массив. Теперь Вы можете во время работы изменять размер массива. Оператор DELETE позволяет удалить элемент массива, а SORT - выполнить его сортировку по алфавиту или по числу. Эти же команды могут быть использованы и при работе со строковыми переменными.

Функция INARRAY выполняет просмотр массива и поиск в нем нужного Вам элемента, то же самое делает INSTRING, но для строковых переменных. Функция LENGTH выдает размер массива и его месторасположение. Она может позволить Вам разделить массив на части и загрузить и обработать его по частям, если он слишком велик, чтобы поместиться в памяти компьютера в то время, как там присутствует БЕТА-БЕЙСИК 3.0. Функции CHAR$ и NUMBER дают возможность создавать "целые" массивы.

Теперь Вы можете редактировать (изменять) переменные в той же мере, как Вы редактируете программные строки. Все программные переменные можно отгружать на ленту единым блоком с помощью SAVE DATA. Форматирование данных можно выполнить с помощью USING или USING$. Функция EOF (End Of File - "конец файла") служит для работы с микродрайвом и может сигнализировать о том, что ввод данных из файла завершен. Оператор SHIFT среди прочих дел выполняет и такую полезную операцию, как изменение регистра букв, которыми записана строковая переменная.

ГЛАВА 6. ГРАФИКА

Используются следующие операторы и функции:

ALTER, CONTROL, CODES, CSIZE, DRAW TO, FILL, GET <Область экрана>, OVER 2, PLOT, POKE, ROLL, SCROLL, WINDOW, XOS/XRG/YOS/YRG, функции SINE, COSE, FILLED, MEMORY$, SCRN$.

Вот в двух словах, для чего они предназначены (более подробно мы рассмотрим каждый оператор и каждую функцию в ближайших выпусках):

ALTER - позволяет гибко управлять цветовыми атрибутами экрана.

DRAW TO - вычерчивание линий к заданной координате.

GET - сохраняет заданную область экрана в виде строковой переменной.

PLOT - восстанавливает на экране (в произвольной области) сохраненный с помощью GET <фрагмент>.

CSIZE - с его помощью Вы можете увеличить или уменьшить размер фрагмента экрана, принятого с помощью GET перед тем, как восстанавливать его по PLOT.

POKE - допускает быстрые манипуляции с областями памяти.

FILL - заполняет область экрана, находящуюся внутри замкнутого контура, избранным цветом INK или PAPER.

ROLL - перемещение экрана или его части в заданном направлении.

SCROLL - то же самое, но с возвратом, когда например то, что ушло за левую границу экрана, начинает появляться справа.

SCRN$ - распознает символы графики пользователя.

WINDOW - организация концепции "окон".

XOS, XRG, YOS, YRG - изменяет исходную координату экрана для графических функций и масштабный коэффициент по двум направлениям.

ГЛАВА 7. СЕРВИСНЫЕ И ОТЛАДОЧНЫЕ ВОЗМОЖНОСТИ

ALTER (…) - поиск и замена по тексту программы.

AUTO - автоматическая нумерация строк.

DEF KEY - этой командой Вы можете задать до 36 операторов, функций или строковых сообщений на пользовательских клавишах.

DELETE - удаление блока программных строк.

LIST/LLIST - распечатка текста программы или ее фрагмента в диапазоне от…до.

LIST/LLIST DATA/VAL/VAL$ распечатка программных переменных.

LIST/LLIST DEF KEY - распечатка определений, присвоенных пользователем назначенным клавишам.

LIST/LLIST PROC - распечатка текста процедуры.

LIST/LLIST REF распечатка только тех строк, в которых есть оператор REF.

REF - поиск программных строк с этим оператором.

RENUM - перенумерация программного блока или копирование.

MEMO - эта функция возвращает доступный объем свободной памяти.

(Продолжение следует)

МИХАЙЛЕНКО В.С.

ЗАЩИТА ПРОГРАММ

Сегодня мы начинаем печатать объемный труд нашего читателя из г. Минска Михайленко В.С., посвященный вопросам защиты компьютерных программ. Книга написана им специально для "ZX-РЕВЮ".

Мы очень рады, что лед тронулся и стали появляться крупные работы отечественных авторов. За те несколько лет, что мы работаем в области информационного обеспечения "Синклер"-совместимых машин, мы очень хорошо узнали, как много у нас талантливейших программистов, способных разработчиков. Но с другой стороны, мы хорошо знаем и то, как трудно их подвинуть на то, чтобы написать хорошую книгу, статью, цикл статей. "Сделаю все, что угодно, но писать книгу - разве что под дулом пистолета", - вот типичный ответ нашего разработчика. Значительно меньшая часть все-таки соглашается, причем охотно, но после первого порыва быстро охладевает и на этом все кончается.

Откуда же мы знаем о существовании этой армии талантов, если они ничего не пишут? - может спросить внимательный читатель. Оказывается, все-таки пишут. Они пишут интересные, хорошо проработанные, наполненные точным, конкретным содержанием критические письма на то, что читают в "ZX-РЕВЮ". Мы благодарны им за это, но увы должны отметить, что собраться и написать пять-шесть страниц, блещущих идеями, нашему разработчику проще, чем собраться с духом и подготовить нормальный законченный материал, освещающий хотя бы одну идею, и который с интересом и пониманием будут читать миллионы начинающих поклонников "Спектрума", а именно такой цифрой мы и оцениваем на сегодняшний день тех, кто уже включился в это всенародное движение.

Итак, мы предоставляем страницы В.С.Михайленко для освещения полезного, как нам кажется, вопроса защиты компьютерных программ. Мы полагаем, что кроме своей утилитарной цели то, о чем он пишет, имеет еще и большое методологическое значение, поскольку приоткрывает перед Вами завесу над многими вопросами, связанными с системными особенностями "Спектрума".

Работа прошла техническое и литературное редактирование "ИНФОРКОМа".

ЧАСТЬ 1

Система защиты компьютерных программ представляет собой исключение возможности просмотра текста программы и внесение в него изменения лицом, не уполномоченным на это автором программы.

Для компьютера "ZX SPECTRUM" существует система мер по защите "Бейсиковских" программ и программ в машинных кодах. Этот цикл статей посвящается защите Бейсик-программ.

Методов защиты программ существует очень много, но среди них можно выделить ряд направлений, по которым можно осуществить подход к данной проблеме:

1. Исключить возможность остановки (прерывания работы программы).
2. Сделать листинг программ нечитаемым.

ГЛАВА 1. Исключение возможности остановки программ.

1.1. Общие рекомендации.

Самое первое, что необходимо сделать - это выполнить программу автостартующей со строки с номером n. Это выполняется при записи программы на ленту командой:

```
SAVE "имя" LINE n
```

Теперь после загрузки программа сама будет стартовать с указанной строки. Тем не менее, она может быть остановлена командой BREAK или по сообщению об ошибке.

Отключить клавишу BREAK можно, если в той строке, с которой происходит автостарт, поместить команду POKE 23613, PEEK 23730-5. Кроме этого, возможно отключение клавиши BREAK за счет использования подпрограммы в машинных кодах, которая будет описана

ниже.

Если же Вы по каким-то причинам не желаете использовать подпрограммы в машинных кодах то необходимо помнить следующее.

Для хорошо отлаженной программы остановку по ошибке можно исключить. Надо знать, что чаще всего при взломе программы сознательно вносят ошибку во время исполнения оператора INPUT. Например, если программа просит от пользователя ввести какое-либо число, он сознательно вводит букву. Программа останавливается с сообщением VARIABLE NOT FOUND (переменной нет). Чтобы избежать этого рекомендуется не использовать оператор INPUT, а организовывать опрос клавиатуры на основе функции INKEY.

Если же Вам необходимо использование оператора INPUT (например при вводе многозначных чисел), то с целью защиты от прерывания можно сделать, чтобы программа либо "зависала" (или самосбрасывалась при остановке), либо вместо сообщения об ошибке осуществляла переход на заранее заданную строку.

Самосброс программы дает использование нижеприведенных команд. Так, если поместить их в строке автостарта, то при попытке сделать BREAK или появлении сообщения об ошибке программа будет сбрасываться:

```
LET ERROR=256*PEEK 23614+PEEK 23613: POKE ERROR, 0: POKE ERROR+1, 0
```

Для обеспечения зависания программы используется размещение в стартовой строке команды:

```
POKE 23659, 0
```

Однако, пользователей компьютера ZX SPECTRUM наверняка заинтересует напомненный им еще раз факт: ни в одной фирменной программе (имеется в виду рабочая программа, а не загрузчик) не происходит сбрасывания или зависания при нажатии на клавишу BREAK. Если после длительной загрузки программа зависнет или сбросится, то это будет иметь недружественный эффект. Гораздо эффективнее в этом плане использование специальных программ в машинных кодах, которые вместо выдачи сообщения об ошибке осуществляют переход на заранее заданный шаг программы.

Эти программы и методы их использования описаны в следующей статье.

1.2 Подпрограмма обработки сообщений с кодами D, H, L.

Нажатие клавиши BREAK во время исполнения программы (или же ей аналогичное STOP IN INPUT) в обычном режиме может приводить к одному из следующих сообщений:

```
D: BREAK - CONT REPEATS
```

Клавиша BREAK нажата во время действия периферийной операции. Действие CONTINUE после этого оператора обычное, т.е. то, что указано в операторе.

```
H: STOP IN INPUT
```

Некоторые введенные данные начинаются с оператора STOP, или была нажата INPUT LINE.

Действие CONTINUE обычное.

```
L: BREAK INTO PROGRAM (BREAK во время исполнения программы)
```

Нажата клавиша BREAK; это было обнаружено между исполнением двух операторов. Строка и номер оператора в строке указывают на оператор, выполненный перед нажатием BREAK, но CONTINUE переходит к следующему оператору.

Чтобы во время нажатия клавиши BREAK остановки не произошло, может быть использована, например подпрограмма в машинных кодах ON BREAK GO TO (подпрограмма N66 из инструментального пакета SUPERCODE 3.5).

Если Вы запустите эту подпрограмму со строки автостарта, то типы остановок D,H,L будут игнорироваться, а программа будет переходить к заранее заданному Вами номеру строки (Первоначально задана строка 9495).

Для тех, кто не располагает пакетом SUPERCODE 3.5, мы приводим дисассемблер данной процедуры.

```
                ORG 60899
60899 CD7C00    CALL 0124
60902 3B        DEC SP
```

```
60903 3B          DEC SP
60904 E1          POP HL
60905 010F00      LD BC,15
60908 09          ADD HL,BC
60909 EB          EX DE,HL
60910 2A3D5C      LD HL,(23613)
60913 73          LD (HL),E
60914 23          INC HL
60915 72          LD (HL),D
60916 C9          RET
60917 76          HALT
60918 CD8E02      CALL 654
60921 7B          LD A,E
60922 FEFF        CP 255
60924 20F8        JR NZ,60918
60926 3A3A5C      LD A,(23610)
60929 FE0C        CD 12
60931 280A        JR Z,60943
60933 FE10        CP 16
60935 2806        JR Z,60943
60937 FE14        CP 20
60939 2802        JR Z,60943
60941 1819        JR 60968
60943 3C          INC A
60944 32815C      LD (23681),A
60947 FD3600FF    LD (IY+0),255
60951 211725      LD HL,9495          ; загрузка номера строки перехода в Бейсике.
60954 22425C      LD (23618),HL
60957 210000      LD HL,0
60960 22445C      LD (23620),HL
60963 3B          DEC SP
60964 3B          DEC SP
60965 C37D1B      JP 7037
60968 C30313      JP 4867
```

Наилучшим вариантом было бы, если бы Вам удалось набрать эту процедуру в Ассемблере и, откомпилировав ее, получить соответствующую программу в кодах. Для тех, кто не имеет такой возможности, привожу программу на Бейсике, которая введет и запустит данную процедуру.

```
5 CLEAR 24999
10 FOR 1=25000 TO 25071
20 READ A: POKE I,A
30 NEXT I
40 DATA 205,124,0,59,59,225,1,15,0,9,235,42,51,92,115,35,114,201,118,205,142,2,
      123,254,255,32,248,58,58,92
50 DATA 254,12,40,10,254,16,40,6,254,20,40,2,24,25,60,50,129,92,253,54,0,255,33,
      23,37,34,66,92,33,0,0,34,68,92,59,59,195,125,27,195,3,19
```

Данная Бейсик-программа позволяет вводить эту подпрограмму в любое место оперативной памяти. Теоретически возможны 3 варианта:

1) Ваша Бейсик-программа сама формирует эту процедуру и запускает ее, т.е. вышеприведенная Бейсик программа является началом Вашей исходной программы, которую необходимо защитить.

2) Вы формируете в удобном месте эту процедуру в кодах, записываете ее на кассету в виде отдельного файла машинных кодов и потом, с помощью программы-загрузчика загружаете эти коды и запускаете.

3) Вы совмещаете программу на Бейсике и эту программу в кодах с использованием оператора REM методом который будет описан в одной из следующих глав и затем сразу после загрузки, непосредственно из Бейсика запускаете эту процедуру.

Третий метод является самым удобным. К недостаткам первого относится то, что пока Бейсик будет формировать подпрограмму в кодах (перебрасывать числовой массив),

взломщик уже сумеет остановить программу. Второй метод усложняет загрузку: в самом деле, Вам придется делать программу загрузчика, загружать коды, запускать их и лишь потом, обезопасив себя от взлома, загружать свою основную программу.

1.3. Подпрограмма обработки сообщений об ошибке, кроме 0:OK; 8:END OF FILE; 9:STOP STATEMENT.

Многие программы имеют возможность ввода ошибки, т.е. останавливаются из-за неправильной своей работы. В этом случае на экране дисплея индицируется сообщение об ошибке. Чтобы этого не происходило, а вместо выдачи сообщений осуществлялся переход к заранее заданной строке Бейсик-программы, используется процедура: ON ERROR GO TO (процедура N 65 из программы SUPERCODE).

Ниже приведена Бейсик-программа, с помощью которой можно загрузить процедуру : "ON ERROR GO TO" в любое место оперативной памяти, в моем примере загрузка осуществляется с адреса 25000.

```
10 CLEAR 24999
20 LET A=25000
30 FOR I=0 TO 72
40 READ B
50 POKE A+I, B
60 NEXT I
70 REM ЗАПУСК ПРОГРАММЫ
80 RANDOMIZE USR 25000
90 DATA 205,124,0,59,59,225,1,15,0,9,235,42,61,92,115,35,114,201,59,59,205,142,2,123,
      254,255,32,248,58,58
100 DATA 92,254,255,40,33,254,1,40,29,254,8,40,25,60,50,129,92,253,54,0,255,33,23,37,
      34,66,92,175,50,68
110 DATA92,253,203,7,254,195,125,27,51,51,195,3,19
```

Процедура "ON ERROR GO TO" имеет длину 73 байта и осуществляет переход к строке Бейсика 9495 (допустимы изменения).

Рассмотрим, для чего же употребляется эта программа. В разделе 1.1. было сказано, что одним из методов взлома является ввод несуществующей переменной вместо числа в операторе INPUT. На уровне Бейсика рекомендовалось использовать опрос клавиатуры с помощью INKEY$. Данная процедура самостоятельно анализирует подобного рода ошибки и осуществляет переход к строке 9495. При этом по адресу 23681 можно узнать код ошибки.

Точно такой же переход эта программа будет осуществлять, если будет произведена попытка деления на 0. Оператор PRINT попытается распечатать символ с помощью управляющего кода AT за пределами экрана; встретится RETURN без GO SUB или NEXT без FOR и др.

Наилучшим способом использования данной процедуры является совмещение ее с Бейсик-программой с использованием оператора REM методом, который будет ниже.

Чтобы подготовиться к его выполнению, Вам необходимо сформировать эту программу в произвольной области ОЗУ, используя Бейсик-программу данного раздела и записать ее на магнитофон в виде отдельного файла CODE.

1.4. Метод защиты, основанный на совмещении Бейсика и программы в машинных кодах.

Многие пользователи компьютера ZX SPECTRUM бывают удивлены, когда после загрузки всего лишь одного Бейсик-файла программы на экране возникают эффекты, которые на Бейсике создать заведомо невозможно. Такое же чувство возникает и тогда, когда после загрузки первого Бейсик файла дальнейшая загрузка ведется необычным способом, несмотря на то, что загружался именно Бейсик-файл (свидетельством этого является то, что Вы подали команду LOAD "").

Все дело в том, что в данном случае использован прием, позволяющий совместить в первом загружающемся файле как Бейсик-команды, так и машинные коды.

Рассмотрим, как это осуществить на практике. Предположим, у нас имеется

отлаженная программа в машинных кодах, имеющая длину n байтов, и мы желаем совместить ее с Бейсик-программой (Данное совмещение накладывает ограничение на программу в кодах: она должна использовать индексную адресацию, переходы могут быть только относительными, т.е. необходимо, чтобы эта программа работала, будучи загруженной в любое место оперативной памяти).

Один из приемов состоит в том, чтобы разместить программу в кодах после оператора REM, т.к. текст, следующий после этого оператора, не обрабатывается компьютером. (Подобное совмещение также накладывает ограничение на использование кодов в программе. Ни один код этой программы не должен быть равен 13(DEC), 0D(HEX), поскольку в Бейсике этот код свидетельствует об окончании строки и символы, следующие за ним, обрабатываются как новая строка Бейсика).

Для того, чтобы осуществить совмещение, нам необходимо сформировать строку с оператором REM так, чтобы количество символов, следующих после REM, было равно числу байтов программы в кодах, т.е. n.

Для этого наберем с клавиатуры:

```
1 REM LLLL...LLLL
```

Мной был использован символ L но в принципе можно использовать к любой другой.

Как Вы уже обратили внимание, строка с оператором REM должна идти первой в программе. Это необходимо для того, чтобы легко можно было определить адрес, по которому нужно грузить машинные коды и из которого вызывать их на исполнение.

Если Вы не изменяли системную переменную PROG, то она указывает на начало Бейсик программы по адресу 23755.

Следовательно, загрузку кодов нам необходимо начинать с адреса 23760 (Это объясняется тем, что первые два кода Бейсик-строки характеризуют ее номер, следующие два - ее длину и пятый символ - это код оператора, в данном случае REM).

Теперь заменим набранные нами символы в количестве n байтов на программу в машинных кодах аналогичной длины. Пододим команду

```
LOAD "" CODE 23760, N
```

(В случае, если системная переменная PROG у Вас смещена, то Вам необходимо узнать адрес начала загрузки кодов, подав следующую команду с клавиатуры:

```
PRINT (PEEK 23635+256 * PEEK 23636+5)
```

Теперь, при загрузке кодов, вместо адреса 23760 укажите адрес, который Вы получили. Загрузив коды, мы добились поставленной цели. Теперь в Бейсике находится программа в кодах, которую при желании можно запустить, подав в одной из нижеследующих строк Бейсик-программы, команду

```
RANDOMIZE USR 23760
```

В случае смещенной системной переменной PROG, Вам будет необходимо видоизменить команду:

```
RANDOMIZE USR (PEEK 23635+256 * PEEK 23636+5)
```

Метод совмещения чаще всего используется, когда Вы хотите защитить наиболее уязвимые места программы, которые Вам наиболее дороги и Вам бы не хотелось, чтобы они были кем-либо изменены. Кроме этого, такой подход позволяет загружать машинные коды без "заголовка" методом, который будет описан далее, что затрудняет вскрытие данного машиннокодового блока, поскольку не известен его адрес.

Как Вы уже обратили внимание, обычно загружающийся программный блок состоит из двух частей. Первая часть является тем заголовком, который содержит название программы, ее длину и адрес, начиная с которого она располагается в памяти компьютера, а вторая часть - это уже непосредственно коды. Следует отметить, что аналогичную структуру имеет и Бейсик программа и массивы, но "заголовки" этих блоков имеют принципиальные различия.

Среди наиболее любопытных особенностей метода скрывания машинного кода в строке REM следует отметить тот факт, что после подачи команды LIST распечатка может быть остановлена с сообщением о неправильном цвете (INVALID COLOR). Это объясняется тем, что в машинных кодах могут встретиться команды, код которых не может быть опознан компьютером в строке Бейсик-программы. И этот любопытный факт, совместно с

использованием метода зануления, который будет описан в следующей главе, дает великолепную защиту от листинга. Необходимо, однако, предупредить читателя, что создавая программы в кодах, для их последующего размещения в Бейсике, необходимо сразу определиться в адресах, т.е. точно установить адреса, где данная программа будет находиться и работать, чтобы не было сбоев из-за неправильной адресации.

Если Вы все же хотите использовать программу в кодах, которая была рассчитана на определенное место в памяти, то чтобы ничего не переделывать, Вам просто необходимо дополнить ее размещенным в начале блоком переноса с использованием команды LDIR (более подробное описание этой команды дано в соответствующих методических разработках "Инфоркома").

1.5. Оригинальный метод защиты, использующийся при загрузке машинных кодов.

Самые первые программы для ZX SPECTRUM отличались примитивностью в плане защиты (да и не только защиты). Обычно они состояли из Бейсик-программы, управляющей загрузкой, которая загружала коды и запускала их с определенного адреса. Взломщик мог беспрепятственно смотреть как Бейсик программу, так и программу в кодах.

Более поздние авторские разработки были гораздо лучше продуманы в отношении защиты.

Как известно, загружающаяся программа состоит из заголовка и самой программы. В заголовке находится описание типа программы: Бейсик, CODE, SCREEN$ и т.д. Кроме этого, там находится название программы и адрес в памяти, с которого начнется загрузка, а также длина программы (для программ в машинных кодах).

Чтобы скрыть эти данные, т.е. длину и адрес начала, разработчики создали систему загрузки, позволяющую загружать машинные коды без заголовка.

Это делается с использованием специальной процедуры в кодах, которая в некоторых случаях совмещается с программой на Бейсике с использованием оператора REM.

К программам, использующим данный метод загрузки, относятся MAT II, RAMBO III и др.

Рассмотрим, как этот прием реализован программе MAT II. Ниже приведен листинг загрузчика.

```
0 REM CRACKED BY MIHAILENKO VS 1991
10 FOR n=0 TO 12: READ a: POKE 64000+n, a
20 NEXT n
30 DATA 55, 62, 255, 221, 33, 0, 64, 17, 254, 27, 195, 86, 5
40 RANDOMIZE USR 64000
```

Фактически Бейсик-программа служит лишь для того, чтобы сформировать и запустить процедуру в машинных кодах. Запуск осуществляется в строке 40, а строки 10...30 ответственны за формирование этой процедуры, начиная с адреса 64000. Ниже приведен дисассемблированный текст данной процедуры.

```
64000 SCF
64001 LD A,255
64003 LD IX,16384
64007 LD DE,7166
64010 JP 1366
```

Действует эта программа следующим образом. В регистр IX загружается адрес начала загрузки кодов, в регистр DE - общая длина программы в кодах, остальная система команд необходима для правильной работы встроенной в ПЗУ программы загрузчика с ленты. Переход на эту программу осуществляется в строке 64010. (Все используемые величины - десятеричные).

Этот тип загрузки используется во многих программах. В частности, программа GREEN BERET использует встроенную процедуру загрузки как подпрограмму, вызывая ее по

CALL 1366 для загрузки нескольких блоков кодов.

Следует добавить, что аналогичным образом действует загрузка в программе RAMBO III. Отличительной ее особенностью является загрузка кодов, занимающих всю оперативную память. Никаким другим способом здесь загрузку осуществить бы не удалось.

1.6. Метод защиты, используемый в программе FIST III.

Защита, используемая в программе FIST III, достаточно проста и не является идеально надежной. Но, в то же время, об этой системе команд мало кто знает и поэтому в некоторых случаях она может оказаться достаточно эффективной.

По нашей классификации этот тип защиты относится к разряду тех, которые делают листинг программы нечитаемым. Для этих целей бордюр окрашивается в черный цвет и подается система команд:

```
BORDER 0: POKE 23624, 0: POKE 23570, 16
```

Этот метод защиты легко можно разблокировать, подав команды:

```
BORDER 7: POKE 23570, 6
```

Кроме защиты, эта программа имеет еще одну особенность, на которую могли не обратить внимание пользователя: заставка в середине экрана формируется на Бейсике. Создана она довольно оригинальным способом, но самым любопытным является то, что системная переменная, используемая в этой программе, в фирменном описании компьютера (автор - Виккерс) охарактеризована, как неиспользующаяся.

Ниже приведем текст программы, после выполнения которой в середине экрана очень крупным шрифтом появляется надпись: GOOD LUCK TO YOU

```
10 REM CRACKED BY MIHAILENKO VADIM MINSK 1991
20 FOR I=72 TO 79
25 POKE 23681, I
30 LPRINT "GOOD LUCK TO YOU"
40 NEXT I
```

В данном случае авторы используют команду вывода на печатающее устройство LPRINT для поэтапного высвечивания на экране вышеуказанной надписи.

Если Вы хотите получить подобное изображение не в середине, а в верхней части экрана, то Вам необходимо подобрать соответствующим образом изменение переменной I.

Любопытный эффект получается при использовании изменения

```
20 FOR I=72 TO 78 и т.д.
```

ГЛАВА 2. Методы защиты от листинга или как сделать текст программы нечитаемым.

Данная глава посвящается изучению методов защиты от листинга, применяемых в большинстве фирменных программ, в частности в их загрузчиках, написанных на Бейсике, либо так или иначе использующих Бейсик. Кроме этого, рассмотрены некоторые методы, применяемые "взломщиками" компьютерных программ, т.н. крэккерами, в частности Билом Гилбертом (Bill Gilbert - если человек с таким именем действительно существует, а не использует псевдоним); PEGAZ SOFTWARE (т.н. корпорация по взлому, которая даже оставляет свой телефон); ROBY CRACKING SERVISE, использующем в своем арсенале достаточно примитивные приемы в отличие от того же Билла Гилберта и пр.

В связи с этим я обращаюсь ко всем читателям данного материала с просьбой не использовать описанные мной приемы взломщиков для установки в созданных не Вами программах Ваших надписей и т.п., поскольку это противоречит элементарным этическим нормам, не говоря уже о порядочности.

После длительной работы я могу сказать с уверенностью, что в компьютере ZX SPECTRUM нельзя создать полностью защищенных программ. Это налагает большую ответственность на человека, описывающего приемы взломщиков. Я ничего не скрывал в изложении данного материала, полагаясь на порядочность и внутреннюю культуру читателей.

Призываю Вас использовать данные приемы исключительно для защиты своих программ, а также для совершенствования своей техники программирования. Вы можете использовать описанные ниже рекомендации для входа в программу с целью изучения новых приемов программирования, но к тем, кто использует, их для того, чтобы калечить чужие программы, не умея написать свои, надо относиться также, как мы отнеслись бы к человеку, который на наших глазах лазит по чужим карманам.

2.1. Общие рекомендации.

Простейший прием сокрытия листинга состоит в том, чтобы сделать одинаковыми цвета символов (INK) и фона (PAPER).

В тех местах, где программа должна сделать вывод на экран в операторе PRINT вставляют в качестве временных правильные цвета.

Например, подав вначале программы команды:

```
10 INK 7: PAPER 7
```

по мере необходимости используем нормальный цвет:

```
20 PRINT INK 0; "ZX SPECTRUM"
```

Другой прием состоит в искажении набора символов путем задания "фальшивого" значения системной переменной CHARS. Для примера наберем с клавиатуры.

```
POKE 23606,8: PRINT "ZX SPECTRUM": POKE 23606,0.
```

* * *

Небольшое отступление:

Краткое описание системной переменной CHARS.

Как известно, системная переменная CHARS ответственна за место расположения шрифта в ПК "SPECTRUM". В своем обычном содержании она указывает на адрес шрифта, зашитого в ПЗУ, а точнее - на тот адрес, который находится на 256 байтов ниже. Это легко проверить:

```
PRINT PEEK 23606
PRINT PEEK 23607
```

Точный адрес можно узнать, подав с клавиатуры команду:

```
PRINT PEEK 23606 + 256*PRINT PEEK 23607+256
```

Вы получите 15616. При этом по адресу 23606 находится 0, а по адресу 23607 - число 60.

Среди практических использований этой системной переменной следует отметить возможность создания своих шрифтов в т.ч. русского и других национальных, а также переключение шрифтов с одного на другой. Подробно эта технология разобрана в разработке "Большие возможности Вашего "СПЕКТРУМа", выпущенной ИНФОРКОМом в 1989 г.

В окончании комментария привожу любопытный пример использования мною этой системной переменной. Многие из Вас, вероятно, видели, что при загрузке некоторых фирменных программ используются счетчики. Они могут быть различны. Либо это постоянно укорачивающаяся линия (уменьшение длины свидетельствует о процессе загрузки, а сама длина показывает непосредственно сколько загрузилось и сколько осталось), использованная, в частности в программе SPLITTING, а также цифровые счетчики, имитирующие механические, подобные тем, что показывают пробег в автомобиле.

Кроме того, такая надпись, имитирующая механический счетчик, использована во время демонстрации в программе FIST III (режим DEMONSTRATION, который появляется после загрузки, если не нажимать никакие клавиши).

Подобный имитатор механического счетчика можно создать и Вам, если использовать возможности системной переменной CHARS.

Наберете предложенную ниже программу и запустите ее:

```
10 FOR I=80 TO 0 STEP -1
20 POKE 23606, I: PAUSE 10
30 PRINT AT 11,15;"0"
40 NEXT I
```

Подобный очень любопытный эффект наблюдается потому, что за каждый цикл

операторов FOR-NEXT процедурой, ответственной за печать, распечатываются 8 байтов предполагаемого знака 0 (в данном случае). Но с каждым проходом системная переменная все ближе подходит к своему истинному значению, а за счет того, что шаблоны цифр в ПЗУ расположены рядом, создается впечатление, что цифры следуют одна за другой по возрастанию или убыванию.

Схематически этот процесс можно представить так:

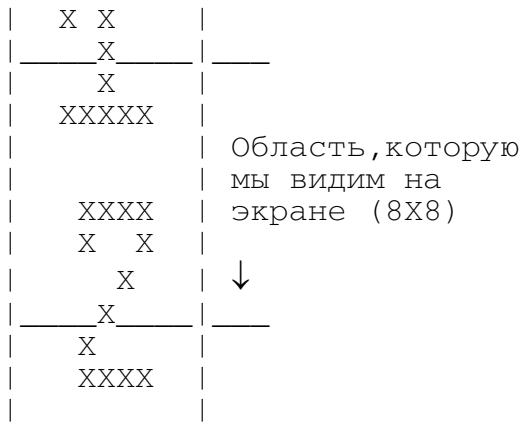


Рис.1

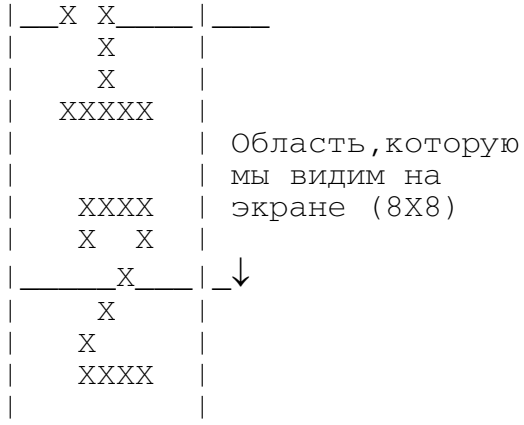


Рис.2

В этой схеме горизонтально расположены байты шаблонов символов, хранящиеся в ПЗУ, и, в зависимости от того, на какой адрес указывает системная CHARS, в области, появляющейся на экране (на экране появляется одно знакоместо 8X8), мы видим опускающиеся вниз цифры, имитирующие движение механического счетчика, до тех пор, пока они не достигнут необходимой нам величины - в моем примере 0.

* * *

Итак, системная переменная CHARS может быть использована для защиты от листинга в Бейсик программах. Если адресовать ее в те области памяти, где вообще ничего нет, например, POKE 23607,200 , то все символы будут выглядеть, как пробелы и на экране вообще ничего не будет. Вам же необходимо перед всяким оператором PRINT включать нормальный режим и выключать его после PRINT.

2.2. Универсальная система защиты - метод нулевых строк.

Почти во всех программах к ZX SPECTRUM присутствует нулевая строка. В большинстве программ она выполняет следующие функции:

1) Защита машинных кодов, размещенных после оператора REM от редактирования.

2) Защита от листинга той части Бейсика, который размещен в нулевых строках.

3) Защита от редактирования надписей, оставляемых фирмами и взломщиками.

Для того, чтобы сделать первую строку программы нулевой, необходимо подать команды:

```
POKE 23755,0
POKE 23756,0
```

Образующаяся строка имеет номер 0. Она реально обрабатывается программой но не поддается редактированию, т.е. ее невозможно вызвать командой EDIT.

Предлагаемая Вашему вниманию универсальная система защиты включает в себя "зануление" всех строк программы (разумеется, этот метод действенен только для программ, где отсутствуют команды условных и безусловных переходов - GO TO, GO SUB и наибольшую эффективность приобретает для небольших программ, управляющих загрузкой). Это становится возможным, т.к. программа выполняет все команды последовательно, а номера строк использует только для команд-переходов. Преимуществ же у этого метода довольно много: при удачном сочетании его с методом защиты от листинга, использующим управляющие коды (подробно описан в следующей статье), это позволяет сделать текст Вашей программы полностью нечитаемым.

Действительно, абсолютное "зануление" не дает возможности указать оператором LIST на какой-либо конкретный шаг в программе. При подаче же команды LIST 0, первая же строка программы с помощью использованных там управляющих кодов сделает одинаковым цвет INK и PAPER, либо закроет текст программы от чтения другим способом.

Мною разработано два метода "зануления". Первый выполняется вручную, с непосредственным Вашим участием. Второй - делает все автоматически, используя специально созданную для этих целей программу. Предпочтительно, на мой взгляд, использовать первый метод, поскольку это позволяет Вам лучше узнать структуру Бейсика и быстрее освоить принципы работы компьютера. Но, тем не менее, в этой статье мною будут рассмотрены оба подхода.

Для того, чтобы правильно произвести "зануление" вручную, необходимо четко представлять себе структуру Бейсик-строки, которую условно можно представить в виде:

MM NN <текст строки> ENTER, где:

MM - номер строки, описан двумя числами;

NN - длина строки, описана тоже двумя числами.

В интерпретаторе Бейсика под номер строки отводятся два знака, причем первым идет старший байт номера, а вторым - младший. (Для тех, кто интересуется, почему соблюдается именно такая последовательность, рекомендую прочитать подробную информацию об этом в трехтомнике "ИНФОРКОМа" Первые шаги в машинных кодах).

Текст строки совпадает с Вашим исходным текстом, за исключением того, что числа представлены в несколько ином виде. (Более подробно об этой и иных системах представления чисел читайте в том же трехтомнике в т. 1 на с. 56). Завершает строку условный код оператора ENTER - 13(тринадцать).

Для того, чтобы сделать номера строк Вашей программы нулевыми, необходимо просмотреть каждую ячейку памяти Бейсика и, зная структуру Бейсик-строки, определить, в каких адресах памяти находится код номера строки, записать номера этих ячеек, после чего с клавиатуры заслать в эти ячейки 0 командой POKE.

Для получения дампинга всех ячеек памяти Бейсика, необходимо использовать небольшую программу, которая размещается в самом конце Бейсика:

```
9997 FOR I=23755 TO 65000
9998 PRINT I; TAB 7; PEEK I; TAB 11; CHR$(PEEK I)
9999 NEXT I
```

Теперь запустим эту программу командой GO TO 9997, после чего на экране будет печататься дампинг Вашей программы (предлагаемые мною строки должны быть набраны после того, как в памяти уже находится Ваша программа) в следующем формате:

- номер ячейки памяти;
- десятичное значение числа, содержащегося там;
- символ, соответствующий данному числу.

В колонке символов, соответствующих содержимому ячеек, Вы увидите приблизительный листинг своей программы. Вашей задачей является обнаружить окончание данной строки Бейсика. Свидетельством этого служит код ENTER - 13. Причем, если десятичное значение 13 распечатается на экране, то символ CHR$ 13 является

непечатным для ZX SPECTRUM и компьютер осуществит переход на следующую строку. Обнаружив, таким образом, номер ячейки, в которой содержится код 13, свидетельствующий об окончании строки Бейсика, Вы записываете номера следующих за ним ячеек памяти, чтобы потом заслать в них 0 командой POKE.

Следует отметить, что Вам необходимо строго следить за тем, чтобы не превратить в 0 номера строк, осуществляющих "дампинг ячеек памяти". После того, как Вы завершите "зануление", Вам необходимо уничтожить строки 9997...9999.

Необходимо подчеркнуть одну маленькую деталь, помогающую облегчить работу с программой распечатки содержимого ячеек. Если эта программа по какой либо причине остановилась с сообщением об ошибке типа INVALID COLOR; NUMBER TOO BIG и т.д., то Вам достаточно набрать с клавиатуры NEXT I и "дампинг" продолжится.

Второй метод заключается в использовании программы для "зануления". Тот процесс, который Вы делали вручную, т.е. анализ строки и "зануление" ее номера, эта программа делает самостоятельно:

```
9990 REM (C) программист Михайленко Вадим Минск, МРТИ, гр 010207 Беларусь, 1991
9991 FOR i=23758 TO 65000
9992 IF PEEK i=13 THEN IF PEEK (i+1)=39 AND PEEK (i+2)=6 THEN STOP
9993 IF PEEK i=13 THEN POKE(i+1),0: POKE (i+2),0: LET i=i+4
9994 NEXT i
```

Эта программа также размещается в конце Бейсик-файла (здесь необходимо строгое соблюдение номеров строк) и осуществляет перенумерацию всех стоящих перед ней строк в 0. После работы эту программу также необходимо уничтожить. Действует же она по следующему принципу:

- в цикле идет анализ Вашей исходной программы, причем при обнаружении конца строки (об этом свидетельствует код символа ENTER - CHR$(13)), номер следующей за ней строки зануляется.

Строка 9992 осуществляет контроль таким образом, чтобы не допустить превращения в 0 (ноль) номеров строк зануляющей программы. (Они находятся по номеру 9990, поэтому наличие этой строки - обязательно). После завершения работы программа останавливается оператором STOP.

Эта программа, однако, не сможет обратить в ноль самую первую строку Вашей исходной программы. Для того, чтобы и ее обратить в ноль, Вам придется подать команду с клавиатуры:

```
POKE 23755,0
POKE 23756,0
```

Теперь Ваша программа защищена: ни одну из ее строк невозможно вызвать для редактирования.

2.3. Использование управляющих кодов.

Одной из малоизвестных и малоисследованных возможностей компьютера является использование в ZX SPECTRUM управляющих кодов. В этом разделе, включающем в себе несколько статей, я опишу применение их для защиты, для использования в Бейсик программах с целью экономии оперативной памяти, а также применение их для создания оригинального хэдера.

Но для начала я постараюсь ввести читателя в курс дела, более подробно ознакомить его с накопленным опытом использования управляющих символов.

* * *

ИНФОРКОМ рекомендует Вам также обратиться к разделу "Маленькие хитрости" в "ZX-РЕВЮ-91", где на с. 116 и 140-142 достаточно подробно разбирался вопрос использования управляющих кодов в операторе PRINT.

* * *

2.3.1. Введение

Не полностью изучив возможности своих персональных компьютеров, многие

пользователи бывают шокированы, когда остановив загрузку, чтобы посмотреть содержание фирменной программы, вместо первой строки Бейсика они видят сплошную строку (без номера) с какой-либо надписью, например:

```
CRACKED BY BILL GILBERT () год
```

Здесь может присутствовать также название программы или надписи иного рода (BILL GILBERT взят для примера - разумеется, другие "хэккеры" тоже используют управляющие коды). Текст же самой программы увидеть не удается.

С помощью управляющих кодов можно смещать текст программы в любом направлении в строке, располагать его в любом знакоместе экрана, управлять цветом символов и фона, регулировать мерцание и создавать повышенную яркость, включать инверсный режим и даже накладывать один символ на другой.

Кроме этого, специально для печати сообщений существует код, позволяющий пропускать сразу 16 символов в строке, т.е. он один как бы заменяет 16 символов типа "пробел".

Знание принципов работы управляющих кодов позволит Вам создать строку, аналогичную описанной в начале, а также так устанавливать цвета, чтобы текста программы не было видно. Кроме этого, управляющие коды можно использовать в "хэдерах" - заголовках файлов - а это позволяет стереть слово PROGRAM при загрузке, либо же расположить название программы в любой точке экрана, либо сделать и то и другое одновременно.

К таким приемам в работе с файлами прибегали многие взломщики фирменных программ (см. EXOLON, GAMEOVER и т.д.).

2.3.2. Самый первый шаг

Рассмотрим формирование строки без видимого номера и каких-либо побочных бейсиковских надписей. Суть этого процесса заключается в том, что с помощью управляющих символов мы смещаем желаемую надпись влево таким образом, чтобы она перекрывала как номер строки, так и бейсиковский оператор (обычно в подобных случаях используется оператор REM).

Изучим конкретный пример. Мы хотим, чтобы после остановки программы и подачи команды LIST у нас сверху появлялась надпись (без номера, указывающего, что это строка Бейсик-программы и без каких-либо операторов):

```
GOOD LUCK TO YOU YOUNG CRACK,
```

что в переводе с английского означает: УДАЧИ ТЕБЕ ЮНЫЙ ВЗЛОМЩИК. Для этого наберем Бейсик-строку:

```
1 REM LLLLLLLLLGOOD LUCK TO YOU YOUNG CRACK
```

Между оператором REM и десятью символами L (символ L может быть заменен любым другим, принципиального значения это не имеет), а также между L и основной надписью Вы не должны делать пробелов. Сейчас мы имеем "полуфабрикат", который будем изменять в соответствии с ранее поставленной целью, для этого четко определим последовательность наших действий:

1) Сначала сместим особым методом исходную надпись на шесть знакомест влево.

2) Зададим цвет INK исходной надписи.

3) Зададим цвет PAPER для исходной надписи.

Будем последовательно выполнять эти пункты:

1. Для того, чтобы сместить надпись на 6 знакомест влево, Вам необходимо заслать управляющие символы оператора "курсор влево" вместо 6 символов L. Для этого наберем с клавиатуры

```
FOR i=23760 TO 23765: POKE i,8: NEXT i
```

Теперь, если Вы без ошибок набрали исходную строку с оператором REM и правильно дали команду с клавиатуры, то при подаче команды LIST Вы должны увидеть на экране надпись, содержащую 4 символа L и основную надпись, но уже без номера строки и без оператора REM.

Выполним второй пункт:

2. Возможно, что защищая свою программу, Вы пожелаете изменить цвета PAPER и INK, но нам необходимо, чтобы в любых условиях Ваша надпись была видна.

Для этих целей необходимо задать цвета непосредственно для надписи. Делается это с помощью прямых команд, засылающих вместо символов L коды управления цветом:

```
POKE 23766,16
POKE 23767,0
```

В ячейку 23766 мы засылаем код управления цветом INK, а в ячейку 23767 непосредственно значение цвета. В данном случае символы будут иметь черный цвет INK.

После этой операции на экране должны быть видны два символа L и следующая за ними основная надпись.

3. Изменим цвет PAPER. Это производится точно таким же способом:

```
POKE 23768,17
POKE 23769,7
```

В данном случае в ячейку с адресом 23768 мы занесли код управления цветом PAPER, а в ячейку 23769 непосредственно числовое значение цвета фона. Теперь надпись, появившаяся на экране после подачи команды LIST, будет содержать только то, что мы желали показать.

А если мы теперь подадим прямые команды

```
INK 0: PAPER 0: BORDER 0: CLS
```

то после подачи команды LIST получим картину, аналогичную той, которую можно наблюдать после остановки многих фирменных программ.

Теперь давайте проанализируем, что же у нас получилось из бывшей изначально достаточно простой Бейсик-строки.

```
23755 ?  0  в этих ячейках
23756 ?  1  распол. ном. строки
23757 ? 44  в этих ячейках
23758 ?  0  распол. длина строки
-----------------------------------
23759  REM  234 здесь наход. код
                оператора REM
-----------------------------------
23760    8 здесь расположен упр.
23761    8 код символа КУРСОР
23762    8 ВЛЕВО
23763    8
23764    8
23765    8
-----------------------------------
23766   16 здесь находятся упр.
23767    0 коды управления
           цветом INK.
-----------------------------------
23768   17 здесь расположены
23769    7 упр. коды, ответственные
           за цвет PAPER
-----------------------------------
23770    здесь расположен наш
. . .    исходный текст: GOOD
23797    LUCK TO YOU YOUNG CRACK
```

Следует предупредить читателя, что для того, чтобы выполнить все вышеизложенное необходимо, чтобы системная переменная PROG, расположенная по адресу 23635-23636 указывала начало Бейсик-строки с адреса 23755. Это будет всегда, если Вы не изменяли его (следует отметить, что это значение может изменяться само при подключении некоторой переферии, в частности интерфейсов внешних устройств, например ZX-INTERFACE-1).

Рассмотрим в качестве следующего примера еще один, достаточно часто используемый хэккерами прием. Например, для того, чтобы сделать дальнейший текст программы (расположенный после строки, которую мы оформили в предыдущем примере) нечитаемым, необходимо сделать так, чтобы компьютер после подачи команды LIST сам останавливал распечатку в необходимом Вам месте. Это можно сделать несколькими способами. В качестве примера рассмотрим один из них.

После того, как компьютер распечатает строку с приветствием для взломщиков, он

должен получить команду (опять-таки с помощью управляющих символов) о том, что задаваемый нами цвет - неправильный. Если такая информация будет получена, то распечатка остановится с выдачей сообщения INVALID COLOR (неправильный цвет).

Для остановки распечатки в необходимом месте, нам понадобится зарезервировать еще 2 символа в том "полуфабрикате", который мы использовали в примере.

Следовательно, строка Бейсик-программы в первоначальном варианте будет иметь вид:

```
1 REM L...L исходный текст LL
      -----
        |
       10 символов
```

Для того, чтобы остановить распечатку, необходимо заменить пару символов LL, расположенных после исходной надписи, на управляющие коды, подобрав их значения таким образом, чтобы они создавали заведомо несуществующую комбинацию цветов.

Чтобы узнать адрес ячейки, в которой расположен последний символ основной надписи, нам необходимо к последнему значению адреса перед этой надписью (в данном случае 25769) прибавить количество символов, содержащихся в данной надписи. В надписи GOOD LUCK TO YOU YOUNG CRACK содержится 28 символов (мы должны считать не только буквы, но и пробелы). Следовательно, теперь мы можем установить адрес ячейки, в которой находится код первой буквы L, первой из тех двух, которыми мы дополнили нашу надпись:

23769+28=23797

Адрес 23797 указывает на последний символ нашей надписи, следовательно, если Вы не делали между основной надписью и парой символов LL пробел. Ячейка, содержащая первое L должна иметь следующий номер:

23797+1=23798

Проверим это, подав команду PRINT CHR$(PEEK 23798).

На экране должен появиться символ L, а для того, чтобы убедиться что мы нашли адрес ячейки именно первого L, а не второго, подадим команду:

```
PRINT CHR$(PEEK 23799)
```

На экране должно тоже появиться L. Теперь нам нужно вместо первого символа L заслать код управления цветом INK:

```
POKE 23796,16
```

После этого компьютер, анализируя данные, поступающие за управляющий символом, теоретически должен будет выполнить следующую команду:

```
INK 76
```

(т.к. код символа L - 76), но это невозможно и компьютер остановит распечатку, выдав сообщение о неправильном цвете:

```
INVALID COLOR
```

Таблица 1

	0	1	2	3	4	5	6	7	8	9
0					TRUE VIDEO	INV VIDEO	PRINT запятая	EDIT	Курсор <--	Курсор -->
10	курсор вниз	курсор вверх	DELETE	ENTER	ЧИСЛО	режим GRAPH	Упр. INK	Упр. PAPER	Упр. FLASH	Упр. BRIGHT
20	Упр. INVERSE	Упр. OVER	Упр. AT	Упр. TAB						

Описание управляющих кодов Спектрума

Таблица 2

Код	Наименование	Назначение
8	BACK SPACE	Код, создаваемый программой ввода с клавиатуры при нажатии "CAPS SHIFT" и "5". Воспринимается большинством принтеров, т.к. это ASCII код перехода назад.
9	RIGHT SPACE	Код, генерируемый программой ввода с клавиатуры при нажатии "CAPS SHIFT" и "8". Код перемещения курсора вправо, но при его использовании позиция печати не меняется. ASCII код табулирования печати строки.
10	DAWN SPACE	Как и вышеперечисленное, но для "CAPS SHIFT" и "7". Это ASCII код заполнения строки.
11	UPSPACE	Как и вышеперечисленное, но для "CAPS SHIFT" и "7". Это ASCII код для смещения печатной позиции вверх.
12	DELETE	Как и вышеперечисленное, но для "CAPS SHIFT" и "7". Это ASCII код заполнения формата.
13	ENTER	Генерируется при нажатии клавиши "ENTER", выполняет возврат каретки и заканчивает печать строки при выводе на принтер. Это ASCII код перевода каретки. Предшествует номеру строки в программе.
14		Используется для чисел в 5-ти байтной форме. Это ASCII код "SHIFT OUT".
15		В стандартном БЕЙСИКе не используется. Это ASCII код "SHIFT IN". Может использоваться в расширениях БЕЙСИКа, например в БЕТА-БЕЙСИКе 3.0.
16	INK CONTROL	Управляющий код INK. Используется перед кодом, содержащим численное значение "INK". Следует отметить, что в качестве кода численного значения цвета используется не ASCII код 2, а значение 2.Например, для изменения цвета печати на экране всех символов на красный, Вам нужно использовать процедуру RST 16 с 16 в регистре A и двойкой.
17	PAPER CONTROL	Управляющий код PAPER. Используется аналогично коду INK. Значения, следующие за ним, соответствуют стандартным значениям цветов "СПЕКТРУМА".
18	FLASH CONTROL	Как и вышеизложенное, но для FLASH. Код может быть соответственно только 0 или 1.
19	BRIGHT CONTROL	Аналогично вышеприведенному для "FLASH", только в данном случае используется для "BRIGHT".
20	INVERSE CONTROL	Как и вышеприведенное, но для "INVERSE".
21	OVER CONTROL	Как и вышеприведенное, но для "OVER".
22	AT CONTROL	Код контроля "AT", который должен сопровождаться номером строки и столбца соответственно.
23	TAB CONTROL	Как и вышеприведенное, но для "TAB". В данном случае код должен сопровождаться только значением столбца.

Если Вы заблаговременно "занулили" строки своей основной программы а также "занулили" первую ее строку, т.е. ту, которая формирует данную надпись, подав команды

```
POKE 23756,0:
POKE 23755,0
```

то теперь, подав команду LIST Вы получите на экране сообщение, которое составляло нашу исходную надпись, а продолжить распечатку никак не сможете т.е. невозможно указать на последующие строки Вашей программы с помощью оператора LIST - все они имеют

нулевой номер.

Команда же LIST 0 снова распечатает строку с исходным сообщением.

В примере 2 для того, чтобы скрыть дальнейший листинг, мы создали заведомо неправильный цвет, но можно поступить иначе, например, сделать после исходной надписи одинаковыми цвета INK и PAPER. Это обеспечит не меньшую надежность в сокрытии листинга. Подадим команды:

```
POKE 23798,16
POKE 23799, 7
```

И, в тоже время, Ваша защита будет иметь серьезный вид, если Вы вообще уберете с экрана какие-либо надписи, т.е. после подачи команды LIST экран будет продолжать оставаться чистым. Это может быть достигнуто в том случае, если мы сделаем одинаковыми цвета символов и фона еще до исходной надписи (строго говоря, необходимость надписи в этом случае отпадает).

Это делается после смещения строки на 6 пикселей влево (см. пример 1 пункт 1) подачей команд:

```
POKE 23766,16
POKE 23767,7
POKE 23768,17
POKE 23769,7
```

На этом наш беглый обзор применения управляющих кодов можно закончить, но в последующих статьях материал будет рассмотрен более детально. Вместе с тем, чтобы Вы начинали не с нуля, рекомендуется проделать описанные эксперименты на компьютере. Это будет способствовать лучшему пониманию материала.

2.3.3. Управляющие коды ZX SPECTRUM.

Управляющие коды в ZX SPECTRUM используются так же, как и в других компьютерах для управления печатью на экране и на принтере. Ниже приведены таблицы 1 и 2.

Внимательно изучив предлагаемый материал, Вы сможете полноценно использовать управляющие коды в своих разработках.

(ПРОДОЛЖЕНИЕ СЛЕДУЕТ)

Дж. Хардман, Э.Хьюзон Перевод Тихоновой Н. А.

40 ЛУЧШИХ ПРОЦЕДУР

Данная книга является сокращенным переводом книги "40 Best Machine Code Routines For The ZX Spectrum With Explanatory Text", J.Hardman & A. Hewson, содержащей в себе набор программ в машинных кодах с весьма подробными разъяснениями принципов их работы.

Значительным сокращениям подверглись те разделы оригинала, в которых рассматриваются вопросы компьютерной терминологии и система команд Z80 - советуем обратиться к трехтомнику по программированию в машинных кодах, выпущенному "Инфоркомом".

При подготовке данного пособия были также исключены описания программ, которые не представляют, на наш взгляд, особого интереса.

РАЗДЕЛ А

1. ВВЕДЕНИЕ

Цель этой книги - обеспечить как начинающего, так и опытного пользователя компьютера ZX SPECTRUM полезными, интересными и развлекательными программами в машинных кодах. Книга имеет 2 раздела.

Раздел А описывает те особенности SPECTRUMа, которые важны при программировании в машинных кодах, некоторые процедуры системного ПЗУ, а также структуру машинного языка.

Раздел В представляет сами программы. Они представлены в стандартном формате, который детально описан в начале раздела. Программы полностью закончены, т.е. они могут быть загружены и использованы индивидуально, без обращений к другим программам.

Предлагаемые программы могут быть загружены с помощью простого загрузчика машинных кодов (MC-LOADER - маш. ЗАГРУЗЧИК), описанного в начале раздела В.

Общие сведения о Бейсике и машинных кодах

Микропроцессор Z80A, на базе которого сделан ZX SPECTRUM, не понимает непосредственно слова БЕЙСИКа. Такие, как PRINT, IF, TAB, и т. д. Вместо этого он выполняет приказы специального языка - своего внутреннего машинного кода. Процедуры ПЗУ, которые придают SPECTRUMу его индивидуальность, написаны на этом специальном языке и состоят из большого количества стандартных подпрограмм для ввода-вывода листинга, интерпретирования и выполнения команд BASICа и др.

Например, стандартные подпрограммы говорят процессору Z80A ("ЧТО ДЕЛАТЬ, ЕСЛИ..."). Если, например, команда BASICа - слово PRINT, то что делать, если следующий элемент имя переменной; или что делать, если следующий элемент - запятая и т. д.

Машинный код состоит из последовательности положительных целых чисел (от 0 до 255), которые диктуют действия для Z80A. Хотя машина использует двоичную форму представления чисел, нет необходимости для человека изучать команды в такой форме. Мы будем использовать десятичную форму, которая обрабатывается МС - ЗАГРУЗЧИКОМ из Раздела В.

Однако даже однообразную строку десятичных чисел трудно интерпретировать и поэтому десятичные числа обычно преобразовываются в специальный язык (ассемблер), который представляет собой определенные аббревиатуры. Язык ассемблера называется так потому, что специальная программа, называемая АССЕМБЛЕРОМ, используется для обработки ("сбора или ассемблирования") команд в машинных кодах при написании (формировании) программы.

Требуется только одно число, чтобы точно определить простую команду Z80A. Например, команда СКОПИРОВАТЬ содержимое регистра С в регистр D - это десятичное число 81 (термин "регистр" более детально описан в главе 3, а пока достаточно если Вы

будете воспринимать C и D, как переменные BASICa). Для таких команд есть точное соответствие между десятичным числом и командой. 81, например, записывается на языке АССЕМБЛЕРа, как LD D,C ("LD", кстати, сокращение слова "load" загрузить). Многие команды ассемблера состоят из подобных простых аббревиатур по этой причине они часто называются мнемониками. Более сложные команды требуют 2, 3 или 4 числа. Но, все равно, для представления их используется одна команда ассемблера. Табл. 1.1. показывает список нескольких чисел, их мнемоник и краткое объяснение действия Z80A.

Таблица 1.1. Некоторые примеры машинных команд Z80A

Ссылка	Десятичное число	Мнемоника	Комментарий
(a)	81	LD D,C	Загрузить в D содержимое C
(b)	14 27	LD C,27	Поместить число 27 в C.
(c)	14 13	LD C,13	Поместить число 13 в C.
(d)	33 27 52	LD HL,13339	Поместить 13339 в пару регистров HL. Обратите внимание: 27+256*52= 13339; 27 поместить в L; 52 поместить в H.
(e)	221 33 27 52	LD IX,13339	Поместить 13339 в пару регистров IX.

Строка (a) таблицы - пример LD D,C рассматривался выше, строки (b) и (c) показывают, как положительное число может быть загружено в регистр (используются два числа: первое определяет действие, которое должно быть выполнено, а второе определяет число, которое должно быть загружено). Строка (d) показывает, как большое целое число может быть загружено в два регистра (H и L) вместе. Здесь второе и третье числа определяют, какие числа должны быть загружены. Последний пример в строке (e) иллюстрирует четырехбайтовый код для загрузки большого целого числа в пару регистров IX. Обратите внимание, что три из четырех чисел такие же, как в строке (d). А дополнительное первое число определяет пару регистров IX вместо HL.

Структура машинного языка объясняется более подробно в главе 3, а полный список мнемоник ассемблера Z80A можно найти в литературе по микропроцессорной технике и программированию.

Итак, наиболее насущный вопрос:
ЗАЧЕМ ИСПОЛЬЗУЮТ МАШИННЫЙ КОД?

На некоторых компьютерах это делается потому, что задачи, которые пользователь желает выполнить, слишком медленно выполняются. В этом отношении ZX SPECTRUM не исключение. Рассмотрим, например, проблему сохранения полного отображения экрана в RAM (ОЗУ) или копирования его обратно на экран с целью создания эффекта мультипликации.

Файл изображения и цветовые атрибуты занимают 6912 байт. Следующая программа BASICa сохранит отображение экрана, но это займет много времени - около 70 секунд:

```
5 CLEAR 58623
10 FOR I=0 TO 6911
20 POKE 58624+I,PEEK (16384+I)
30 NEXT I
```

Причина такой медленной работы в том, что SPECTRUM тратит больше всего времени на декодирование команд BASICa перед их выполнением. Некоторое количество времени также тратится на преобразование чисел в двухбайтную форму (которую понимает Z80A) из десятичных чисел в пятибайтной форме (с которыми оперирует BASIC), а также на выполнение пятибайтовой арифметики.

Так, в нашем примере по переброске экрана должны быть выполнены следующие шаги:

1. Прибавить i к 16384.
2. Преобразовать результат в форму двух байтов.
3. Восстановить содержимое адреса PEEK.

4. Прибавить i к 58624.
5. Преобразовать результат в форму 2-х байтов.
6. Сохранить полученное (PEEK) значение по заданному адресу (POKE).
7. Прибавить 1 к значению I и сохранить результат.
8. Вычесть i из 6911. Если результат положительный, то идти на пункт 1.

Во время прохождения цикла, SPECTRUM должен декодировать каждую команду снова, так как в данном случае память не используется для сохранения последовательности предыдущих действий. Легко увидеть, что компьютер тратит более 99 процентов времени на подготовку к выполнению задачи, а не на выполнение самой задачи. Аналогичная программа в машинных кодах для сохранения экрана выполняется практически мгновенно. Пример такой программы дан в Разделе В.

2. ВНУТРЕННЯЯ СТРУКТУРА ZX SPECTRUM

Компьютер - это машина, которая способна запомнить последовательность команд и затем выполнить их. Конечно, чтобы сделать так, требуется память, в которой команды могут быть сохранены. ZX SPECTRUM имеет два типа памяти. Первый тип - ПЗУ (ROM), в котором содержится фиксированная последовательность инструкций, введенных в машину ее изготовителем.

Второй тип - ОЗУ (RAM). RAM - это "блокнот для записей" компьютера. Когда компьютер выполняет задачу, он непрерывно просматривает, что находится в RAM ("чтение" из памяти) и обновляет содержимое RAM ("запись" в память). SPECTRUM не использует свой "блокнот" для записей случайно. Различные части RAM используются для хранения различных видов информации. Программа BASICa, введенная пользователем, например, хранится в одной части RAM, в то время как переменные, используемые этой программой, хранятся в другом месте. Размер "блокнота" для записей ограничен, и потому машина точна при распределении пространства для информации, которую она хранит. Свободное пространство собрано в одном месте, и, если пользователь хочет добавить строку в свою программу, информация в RAM должна быть "перетасована" по всей длине, используя некоторую свободную область для вставки дополнительной строки.

В значительной степени эта глава посвящена объяснению организации RAM SPECTRUMa, так как многие программы раздела В предназначены для манипуляций с RAM. Глава содержит в себе описание дисплейного файла, атрибутов, буфера принтера, системных переменных, программной области и области программных переменных. В конце раздела описываются стандартные подпрограммы из ROM, к которым ссылаются программы в Разделе В.

Карта памяти

RAM имеет 49152 ячейки памяти. Каждая ячейка может хранить одиночное целое число от 0 до 255 включительно и задается адресом, который является положительным целым числом от 0 до 65535. Адреса от 0 до 16383 зафиксированы для постоянной памяти - ROM. Первый адрес RAM (ОЗУ) - 16384. Табл. 2. 1. - упрощенная карта памяти SPECTRUMa, которая показывает, как используется RAM с адреса 16384.

Дисплейный файл, который хранит отображаемую на экране информацию, занимает ячейки от 16384 до 22527. Атрибуты, которые определяют цвет, яркость и т.д. для каждого знакоместа экрана следуют непосредственно дальше: в ячейках 22528 - 23295.

Первые 5 адресов в колонке Таблицы 2.1. являются фиксированными, т.к. дисплейный файл, атрибуты и т.д. занимают фиксированное положение. Пятая область назначена для карты памяти микродрайва. Это небольшое периферийное устройство представляет собой нечто среднее между магнитофоном и дисководом. Грубо говоря - это дешевая альтернатива дисководу. Скорость его работы достигаете за счет того, что, с одной стороны, лента движется с очень высокой скоростью, а с другой - за счет организации прямого доступ к файлам, как на диске, а не последовательного, как на магнитофонной кассете. Вот для того чтобы иметь в некоем месте хранилище с данными о том, что записано на каждом секторе микродрайва в памяти компьютера и выделен область карты памяти

микродрайва. Если микродрайв (а точнее INTERFACE-1, через который он подключается), подсоединен к SPECTRUMу, эта область содержит информацию о его секторах. Если же не подсоединен, эта область не нужна и в этом случае шестая область (информация о каналах) размещена непосредственно за четвертой областью, системными переменными, т.е. стартовый адрес области информации о каналах и всех последующих областей не фиксирован, а может "плавать" вверх-вниз в RAM. SPECTRUM хранит стартовый адрес всех этих областей в системных переменных. Область системных переменных находится перед картой микродрайва в ячейках 23552-23733 включительно. Эти адреса являются все время строго фиксированными.

Стартовый адрес или имя системной переменной	Ячейка системной переменной	Содержимое памяти
16384	-	Дисплейный файл
22528	-	Атрибуты
23296	-	Буфер принтера
23552	-	Системные переменные
23734	-	Карта микродрайва
CHANS	23631	Область информации о каналах
PROG	23635	Адрес начала программы на БЕЙСИКе
VARS	23627	Адрес начала области программных переменных.
E-LINE	23641	Адрес области редактирования
WORKSP	23649	Буфер INPUT
STKBOT	23651	Стек калькулятора
STKEND	23653	Свободная область
SP	-	Машинный стек и стек GO SUB
RAMTOP	23730	Пользовательские подпрограммы в машинном коде.
UDG	23675	Графика пользователя.
P-AMT	23732	Физическая вершина ОЗУ.
Таблица 2.1. Карта памяти. Указатель стека SP хранится не в RAM, а в SP -регистре микропроцессора Z80A.		

Адреса ячеек, которые хранят стартовые адреса всех "плавающих" областей, даны в колонке 2 таблицы 2.1. Адрес области начала программы BASICa, например, хранится в ячейках 23635 и 23636 в области системных переменных.

Примечание: Ссылка к системной переменной с помощью адреса, по которому она хранится, довольно неудобна. По этой причине в Таблице 2.1. в 1-ой колонке даны условные имена системных переменных. Эти имена удобны только для пользователей, в то время как SPECTRUMOM они, естественно, не распознаются. Например, введенная строка:

```
PRINT PROG
```

даст сообщение об ошибке

```
"2: variable not found" ("Переменная не найдена").
```

PEEK и POKE

Карта памяти - это ключ для понимания того, как компьютером используется RAM-память. Для непосредственного управления RAM используются ключевые слова BASICa - PEEK и POKE, которые позволяют просмотреть и изменить содержимое любой ячейки памяти. PEEK - функция вида:

```
PEEK <адрес>
```

Адрес - это целое положительное число от 0 до 65535 или арифметическое выражение, которое, выполняясь, дает положительное число. Важно заключить арифметическое выражение в скобки, т.к. PEEK 16384 + 2 интерпретируется, как (2 + результат PEEK 16384), тогда как PEEK (16384 + 2) интерпретируется, как PEEK 16386

Значение, выдаваемое функцией PEEK - это число, хранящееся по данному адресу в текущий момент. Оно всегда будет положительным от 0 до 255.

Выше объяснялось, что системная переменная PROG хранится по адресу 23635, но значение PROG (т.е. адрес в RAM) всегда больше числа 255. Следовательно, для его хранения необходимы две смежных ячейки с адресами 23635 и 23636. Значение PROG может быть получено с помощью командной строки:

```
PRINT "PROG="; PEEK 23635 + 256 * PEEK 23636
```

Все адреса хранятся в 2-х смежных ячейках в такой форме и могут быть получены вводом: PRINT PEEK 1-я ячейка + 256 * PEEK 2-я ячейка

Например, если SPECTRUM используется без подсоединенного микродрайва, область карты микродрайва не будет существовать, и информация о каналах будет располагаться непосредственно после области системных переменных, т.о. системная переменная CHANS должна быть такой же, как стартовый адрес карты микродрайва, когда он существует, т.е. 23734. CHANS хранится в 23631 и 23632 и, следовательно, после ввода: PRINT PEEK 23631 + 256 * PEEK 23632 будет получено значение 23734.

Функция PEEK может быть использована также для просмотра содержимого любой из ячеек ПЗУ. Это очень полезно. Просмотр любой ячейки не приводит к разрушению или искажению программы или переменных. Иногда результаты PEEK могут быть обманчивыми, т.к. содержимое ячейки, которая просматривалась, может изменяться в течение или непосредственно после выполнения команды просмотра.

Например, если просматривается содержимое ячеек, которые связаны с левым верхним углом экрана дисплея, то результаты, распечатанные в верхнем левом углу экрана, будут уже устаревшими, т.к. в момент вывода изображения мы уже изменили эти ячейки.

Команда POKE более рискованная, чем функция PEEK, т.к. задавая ее, пользователь вмешивается в функционирование компьютера. Использование этой команды может быть причиной сбоя машины или ее останова.

Формат команды:

POKE <адрес>,<число>

Адрес - это положительное целое число от 0 до 65535 включительно или арифметическое выражение, которое дает такое число. В этом случае нет необходимости заключать арифметическое выражение в скобки, т.к. POKE - это команда, а не функция (хотя есть негласная заповедь, что лишними скобками программу не испортишь). Число, помещаемое в ячейку, может быть от 0 до 255 включительно.

Дисплейный файл

Обычно дисплей содержит 24 строки по 32 символа. Дисплейный файл занимает ячейки от 16384 до 22527, т.е. 6144 ячейки в общей сложности. Следовательно, количество ячеек, используемое для символа:

6144/(24*32)=8.

Эти 8 ячеек формируют изображение символа экране, называемое знакоместом.

Наиболее легкий путь получения общего впечатления о том, как организован дисплейный файл - это печать (PRINT) картинки на экране, сброс (SAVE) экрана на ленту, очистка экрана (CLS) и загрузка (LOAD) картинки экрана вновь. Программа P2.1 сохраняет (SAVE) и загружает (LOAD) экран, используя графический символ на клавише 5 для создания оригинальной картинки.

Программа P2.1.

```
100 FOR  I=0 TO 703
110 PRINT " "; : REM символ, находящийся на клавише "5" в G-режиме
120 NEXT I
130 SAVE  "Picture"  SCREEN$
140 CLS
150 INPUT "Перемотайте ленту и включите воспроизведение"; z$
160 LOAD  "Picture"  SCREEN$
```

Когда картинка загружается с ленты, видно, что дисплей разделен на три зоны по 8 символьных строк в каждой, а каждая строка разделяется на восемь пиксельных линий. SPECTRUM загружает сначала верхние пиксельные линии для первых восьми строк, затем следующие пиксельные линии тех же восьми строк и т.д. Таким же образом формируются средняя и нижняя части дисплея.

Другой путь понимания формирования дисплея - это рассмотреть, где хранятся 8 байтов, которые используются для формирования символа в верхнем левом углу экрана. Первый байт формирует самую верхнюю 1/8 часть символа и размещается в начале дисплейного файла по адресу 16384.

Команда POKE 16364,0 очистит верхнюю линию пикселей самого верхнего левого знакоместа, в то время, как POKE 16384,255 закрасит всю эту линию. При помещении в эту ячейку числа от 0 до 255 мы получим в этом месте экрана различные штрихи. Вторая сверху линия в первом знакоместе на экране не сформирована числом, хранящимся в ячейке 16385, - эта ячейка используется для верхней линии пикселей в соседнем символе. Вторая линия сверху в первом знакоместе формируется числом, хранящимся в ячейке 16384+32*8=16640.

Подобным же образом вычисляются адреса оставшихся шести линий этого знакоместа.

Следовательно, образ символа в верхнем левом углу экрана определяется содержимым адресов 16384, 16640, 16896, 17152, 17408, 17664, 17920, 18176.

Программа P2.2. позволяет Вам экспериментировать, помещая различные числа в эти восемь ячеек.

Программа P2.2. Программа для создания символа в верхнем левом углу экрана.

```
10 REM подпрограмма установки символа в верхнем углу экрана
20 INPUT "Символ состоит из восьми байтов, каждый из которых - число в диапазоне от 0 до
      255. Введите номер байта (от 0 до 7)";n
30 IF n<0 OR  n>7 OR  n<>INT n THEN BEEP  0.2,24: GO TO 20
40 INPUT "Ввести содержимое байта";m
50 IF m<0 OR m>255 OR m<>INT m THEN BEEP  0.2,24: GO TO 40
60 POKE 16384+8*32*n,m
```

Программа P2.3. Программа декодирования атрибута.

```
10 REM декодер атрибутов
20 DATA "Black", "Blue", "Red", "Magenta", "Green", "Cyan", "Yellow", "White", "Bright",
      "Flash"
30 DIM C$(8,7)
40 FOR  I=1 TO 8
50 READ  C$(I)
60 NEXT I
100 REM Декодер атрибутов
110 INPUT "Ввести число от 0 до 255. Эта программа декодирования интерпретирует его в
      файл атрибутов";n
120 IF  n<0 OR  n>255 OR  n<>INT n THEN BEEP  .2,24: GO TO 110
200 PRINT "Цвет символа - "; c$(1+n-8*INT(n/8))
210 PRINT "Цвет фона - "; C$(1+INT(n/8)-8*INT(n/64))
220 IF INT(n/64)=1 OR INT(n/64)=3 THEN PRINT "СИМВОЛ - BRIGHT"
230 IF n>127 THEN PRINT  "Символ будет мерцать (FLASH)"
300 PRINT AT 6,0; "::::::::::::::::::::::::::::::"
310 FOR i=22720 TO 22751
320 POKE i,n
330 NEXT i
500 INPUT "Для повторения - ENTER";z$
510 CLS
520 GO TO 110
```

Каждая ячейка в дисплейном файле определяет восемь пикселей на экране. При этом

число, которое хранится в данной ячейке, преобразуется в двоичную форму и затем устанавливаются восемь пикселей, соответственно двоичным цифрам. Например, 240 после преобразования в двоичное число дает:

11110000

Следовательно, если ячейка содержит число 240, четыре из восьми пикселей, соответствующие единицам, будут высвечены, а оставшиеся нет. Следовательно, дисплейный файл состоит из 6144 ячеек памяти по 8 ячеек для одного знакоместа.

Каждая ячейка определяет горизонтальную полосу из восьми пикселей. Ячейки, относящиеся к данному знакоместу, не расположены последовательно одна за другой.

Атрибуты

Содержимое дисплейного файла определяет, какие пиксели высвечиваются на экране, цвет фона (PAPER), символа (INK), яркость (BRIGHT) и мерцание (FLASH) определяются с помощью атрибутов. Область атрибутов занимает ячейки 22528 - 23295 - по одной ячейке для каждого из 768 знакомест.

Соответствие между содержимым ячеек памяти файла атрибутов и самими атрибутами - следующее:

Значение атрибута = 128*FLASH + 64*BRIGHT + 8*PAPER + INK

FLASH и BRIGHT принимают значение 1, если соответствующее условие установлено, а PAPER и INK принимают значение требуемого цвета, как показано на клавиатуре (красный, например 2).

Программа P2.3. декодирует атрибуты, т.е. данное значение атрибута распечатывает с соответствующим цветом PAPER и INK.

Буфер принтера

256 ячеек ОЗУ, следующие за областью атрибутов, используются, как буфер для хранения строки символов, которая должна быть передана на принтер.

Многие программы в Разделе В будут использовать буфер принтера для передачи данных BASICa или от клавиатуры в подпрограммы. Буфер подходит для этой цели, т.к. его ячейки фиксированы и маловероятно, что пользователь пожелает его использовать для других целей.

Единственно важное ограничение в этом случае - не использовать команды BASICa, которые требуют работы с принтером - LLIST, LPRINT, COPY.

Есть еще и ограничение для владельцев 128-килобайтных машин. У них нет буфера принтера. Дело в том, что буфер принтера предназначался в оригинальной модели для поддержки недорогого специализированного узкопечатного ZX-принтера. В фирменной модели 128-килобайтных машин есть порт подключения полноценного матричного принтера, обладающего собственным буфером и необходимость в этом буфере отпала, зато в этих моделях необходимо больше системных переменных и под них отдали область буфера ZX-принтера. Теперь, если в режиме 128K пользователь что-либо зашлет в эту область, то нарушив системные переменные он выведет программу из строя.

Область программ на BASICe

Обычно эта область начинается с адреса 23755 и на нее указывает содержимое системной переменной PROG (23635,23636), но есть и исключения за счет некоторых видов периферийных устройств.

Если, например, к компьютеру подсоединен микродрайв, то начало этой области сдвигается, в этом самом общем случае начало области программ на BASICe и определяют с помощью системной переменной PROG. Ниже предполагается, что такая периферия не подсоединена.

Программа P2.4. распечатывает содержимое 18-ти ячеек в начале программной области, как показано на рис. 2.1. В этих 18 ячейках хранится первая строка данной программы.

```
10 REM Peek program
```

```
20 FOR i=23755 TO 23772
30 PRINT i,PEEK i
40 NEXT i
```

Программа P2.4. Программа для просмотра содержимого первых 18-ти ячеек в программной области.

23755	0
23756	10
23757	14
23758	0
23759	234
23760	80
23761	101
23762	101
23763	107
23764	32
23765	112
23766	114
23767	111
23768	103
23769	114
23770	97
23771	109
23772	13

Рис.2.1. Форма, в которой строка 10 REM peek program хранится в программной области.

Номер строки (10) хранится в первых двух ячейках в форме:

Номер строки = 256*PEEK первый адрес + PEEK 2-й адрес.

Следующие 2 ячейки: 23757 и 23758 хранят длину оставшейся части строки, начинающейся в ячейке 23759. В нашем случае: 14+256*0=14

Т.о. следующая строка начинается с ячейки:

23759+14=23773

Ячейка 23759 хранит в себе число 234, которое является кодом ключевого слова (токена) REM. Следующие 12 ячеек хранят коды символов одиннадцати букв и пробела, составляющих фразу Peek program. Последняя ячейка хранит число 13, которое является кодом для ENTER, определяющим конец строки

Таблица 2.2. показывает метод кодирования программы в программной области.

Ячейки	Номер строки
1 и 2	Номер строки
3 и 4	Длина строки, исключая первые 4 ячейки
5	Код команды
Конец	Символ ENTER, число 13

Табл. 2.2. Этот метод используется для кодирования строк программы.

Момент, который опущен в таблице - это описание метода хранения числовых значений, имеющих место в программе. Этот метод может быть исследован на примере строки:

```
10 LET a=1443
```

Введите ее в программу P2.4. На рис.2.2 показан результат работы программы в этом случае.

23755	0
23756	10
23757	14
23758	0
23759	241
23760	97
23761	61
23762	49
23763	52
23764	52
23765	51
23766	54
23767	0
23768	0
23769	163
23770	5
23771	0
23772	13

Рис. 2.2. Формат, в котором строка 10 LET a=1443 хранится в программной области.

Ячейки 23755-23758 такие же, как в прошлом примере. Затем следуют коды для LET, a, =, и четыре кода цифр, которые вместе формируют число 1443. Следующий элемент, находящийся в ячейке 23766, - это код 14. Этот код указывает, что следующие 5 ячеек хранят число в специальном пятибайтном формате. Линия заканчивается в ячейке 23772 вводом символа ENTER, как и ранее.

* * *

Примечание ИНФОРКОМа. Получается так, что в одной строке число как бы записано дважды. Первый раз своими символами, а второй раз - в пятибайтной форме после кода 14. Зачем это нужно?

Дело в том, что первое представление используется для того, чтобы БЕЙСИК-интерпретатор знал, что вам показать на экране, а второе - используется для расчетов во внутреннем калькуляторе компьютера. (О калькуляторе читайте в т. 1 и 2 нашего трехтомника).

Самое интересное, что они могут и не совпадать. В этом случае Вы на экране будете видеть одно, а программа будет обрабатывать совсем другое число, и этим нередко пользуются в защите программ. Например, Вы видите на экране LET a=1257: GO TO a и пытаетесь проследить работу программы, а на самом деле там было записано нечто совсем другое и компьютер делает переход не к строке 1257, а туда, куда надо.

Желающие могут посмотреть загрузчик программы BOMBJACK. В череде относительно несложных вывертов этот там стоит одним из первых. Но если на него "клюнуть", атака на программу никак не получится.

* * *

Цифровой пятибайтный формат

Пять ячеек памяти используются для хранения чисел в программе на BASICe (исключая номера строк). Целые числа в диапазоне от -65535 до +65535 хранятся таким же образом, как в формате Z80A. Для этих чисел первые две ячейки и последняя содержат 0, а третья и четвертая хранят число в двухбайтной форме:

Число = PEEK 3-я ячейка + 256* PEEK четвертая ячейка

Таким образом 16533 хранится в пяти ячейках как

0 0 169 64 0

потому, что

169+256*64=16553

Дробные числа хранятся в формате с плавающей запятой таким образом: экспонента в первой ячейке, а мантисса в следующих четырех ячейках, т.е.:

число = мантисса * 2^экспонента

Первая ячейка мантиссы используется также для определения знака числа. Если ячейка содержит значение в пределах от 0 до 127, число положительное, если нет - отрицательное.

Программа P2.5 может быть использована для восстановления дробного числа из составляющих его пяти компонентов.

```
10 PRINT "Ввести экспоненту и четыре байта мантиссы. Все числа должны находиться между 0
      и 255 включительно."
20 INPUT e,a,b,c,d
30 PRINT ,, " Exponent= ";e
40 PRINT "Mantissa=  "; a,,b,,c,,d
50 PRINT ,,"The number=  "; (2*(a<128)-1)*2^(e-160)*
      (((256*(a+128*(a<128))+b)*256+c)*256+d)
```

Программа P2.5. Эта программа восстанавливает дробное число.

Область переменных

Область переменных начинается в ячейке, адрес которой хранится в системной переменной VARS (ячейка 23627). Как бы ни была объявлена новая переменная, т.е. в программе или непосредственным вводом с клавиатуры, для нее резервируется соответствующее количество свободного пространства в этой области.

Все имена переменных начинаются с буквы. Различий между верхним и нижним регистром нет. Эти ограничения позволяют SPECTRUMу манипулировать с кодом первого символа каждой переменной и, таким образом, он может различить шесть типов переменных просмотром диапазона, в котором находится код.

Все цифровые переменные с односимвольными именами, например, имеют коды в пределах от 97 до 122; буква a - 97; b - 98; c - 99 и т.д. Подобным же образом цифровые массивы имеют коды в пределах 129 – 153, т.е. a - 129; b - 130; c - 131 и т.д. Диапазоны кодов представлены в таблице 2.3.

Длина каждого типа переменной также показана в таблице 2.3.

Тип переменной	Диапазон символьного кода	Длина в области переменных
Цифровой (односимвольное имя)	97 - 122	6
Цифровой (многосимвольное имя)	161 - 186	5 + длина имени
Цифровой массив	129 - 154	4 + 2 * размерность + 5 * общее количество элементов.
Управляющая переменная цикла FOR-NEXT	225 - 250	18
Символьная строка	65 - 90	3 + длина строки
Символьный массив	193 - 218	4 + 2 * размерность + общее число элементов.
Таблица 2.3. Переменные. Диапазон кодов и длина переменных.		

Подпрограммы ПЗУ

Некоторые из представленных в книге программ (раздел B) используют стандартные подпрограммы ПЗУ:

RST 16 - распечатывает содержимое аккумулятора.

CALL 3976 - вставляет символ, хранящийся в аккумуляторе, по адресу, хранящемуся в паре регистров HL.

CALL 6326 - если аккумулятор хранит код 14, устанавливается нулевой флаг и увеличивается пара регистров HL в пять раз.

CALL 6510 - возврат в HL адреса в ОЗУ той строки, номер которой был передан в эту подпрограмму через HL.

3. МАШИННЫЙ ЯЗЫК Z80

Регистры Z80A

Во время выполнения программы компьютер не обновляет непосредственно содержимое памяти. Он копирует содержимое ячейки памяти в регистр и оперирует содержимым регистра. Регистры в машинном языке имеют функцию, схожую с переменными в BASICe, т.е. используются для хранения чисел. Они отличаются от переменных BASICa тем, что число их ограничено и они существуют в процессоре, а не в RAM. Кроме того, они могут хранить только один байт, или 2 байта в паре регистров.

Z80A - имеет несколько регистров и потому может хранить несколько чисел одновременно. Благодаря этому снижается время при обмене информацией между процессором и памятью.

Регистр "A" (Аккумулятор)

Аккумулятор - это наиболее важный регистр, т.к. больше всего арифметических и логических команд выполняется с содержимым этого регистра. Этот регистр называется аккумулятором, т.к. результат последовательных операций накапливается ("аккумулируется") в нем. Некоторые команды, которые обращаются к аккумулятору, используют другой регистр или адрес памяти в качестве источника данных. Например, инструкция

```
ADD A,B
```

дает процессору команду прибавить содержимое регистра B к содержимому регистра A, поместив результат в A.

Флаг	Мнемоника		Использование
Знак	N	P	Включается, когда результат последней операции отрицательный.
Нуль	Z	NZ	Включается, когда результат последней операции равен нулю или имело место совпадение.
Перенос	C	NC	Включается, когда происходит переполнение регистра, т. е. последний результат больше того, что может быть записанным в один байт (или в два байта для операций с парой регистров).
Четность/ Переполнение	PE	PO	Флаг включается, если в байте результата предыдущей операции количество включенных битов есть величина четная. В некоторых операциях этот флаг свидетельствует о переполнении.

Таблица 3. 2. Четыре флага, которые контролируют наибольшее количество операции Z80A.

Флаг-регистр "F" (регистр состояний)

Регистр F довольно значительно отличается от всех остальных, т.к. его содержимое не рассматривают как один байт, а рассматривают как 8 индивидуальных битов, что конечно же одно и то же. Эти биты используются в качестве так называемых флагов для управления последовательностью выполнения программы. Каждый флаг используется для определения того, какое из двух логически противоположных условий имело место при выполнении предыдущей операции. Например, флаг нуля определяет, был ли равен нулю результат последней операции сложения, вычитания и т. п. Только 4 из восьми флагов наиболее интересны для пользователей, их свойства кратко изложены в Таблице 3.2.

Флаг знака наиболее простой. Кроме того, условленно, что в операциях со знаковыми числами седьмой бит (старший бит в байте) тоже используется для хранения знака. Он

включается, когда число отрицательно, иначе снимается.

Этот флаг отражает знак последнего результата.

Флаг нуля устанавливается, если результат последней операции равен нулю. Он также используется инструкциями сравнения, которые фактически аналогичны вычитанию, при котором результат игнорируется.

Флаг переноса регистрирует переполнение, которое имеет место, если результат сложения длиннее чем то, что может быть записано в регистр, либо имеет место заем при вычитании. Имеется также несколько инструкций сдвига, в которых биты в регистре А сдвигаются влево или вправо циклически через флаг переноса.

Флаг четности/переполнения - это два флага в одном. Он используется, как флаг переполнения при выполнении арифметических операций для определения, был ли бит 7 получен в результате переноса или сгенерирован битом 6 при "взятии займа". Это используется в случае, если бит знака был искажен. Логические инструкции используют тот же самый флаг для определения четности результата.

Примечание: четность двоичного числа - это четность количества битов, установленных в единицу. Если количество четно, то говорят, что имеет место четность, если оно нечетно, то говорят, что имеет место нечетность. Флаг устанавливается, если имеет место четность.

Результат некоторых инструкций зависит от текущих установок отдельных флагов, например, инструкция

```
JR Z, D
```

позволяет Z80A "перепрыгнуть" через несколько инструкций, если флаг нуля установлен (т. е. сделать условный переход типа IF...THEN GO TO). Если флаг нуля не установлен, процессор выполняет следующую инструкцию в обычной последовательности, т.о. флаговый регистр очень важен, т.к. он позволяет процессору принимать решения и переходить к другой части программы.

Регистры счетчики "B" и "C"

Регистр В и, в некоторой степени, регистр С, с которым он может использоваться в паре, доступен для нескольких применений. Но наиболее важно использование его в качестве счетчика. Мы уже видели, как выполнение программы может управляться использованием флага 0 в инструкции JR z, n.

Другая инструкция: DJNZ n использует флаг 0 для создания циклов, используя регистр В в качестве счетчика аналогично циклам FOR-NEXT в BASICe. Когда встречается эта инструкция, Z80A уменьшает содержимое регистра В на 1. Если результат равен нулю, то выполняется следующая инструкция в нормальной последовательности, если результат не равен нулю, подпрограмма "перескакивает" n инструкций. Если программист использует отрицательное значение для n, "прыжок" осуществляется в программе назад и, если здесь нет других переходов, процессор, конце концов, встретит ту же самую инструкцию вновь.

Т.о. загрузкой регистра В определенным числом и установкой соответствующего смещения n определенная часть кодов может быть выполнена заданное количество раз. Регистр В содержит только один байт и поэтому может устанавливаться на любое число от 0 до 255, т.е. используя описанный механизм через определенный блок кодов может быть сделано максимум 255 проходов. Для осуществления более 255 проходов в цикле нет эквивалентных инструкций, но имеются очень мощные инструкции, которые используют все 16 битов регистровой пары BC, как счетчик с максимальным значением 65535. Например, инструкция CPDR. Когда Z80A выполняет ее, то:

1) Содержимое BC уменьшается на 1;

2) Уменьшается содержимое HL (о регистровой паре HL см. ниже);

3) Сравнивается содержимое аккумулятора А с содержимым ячейки памяти, адрес которой находится в паре регистров HL.

Процессор повторяет эти действия до тех пор, пока не будет совпадения между содержимым А и ячейки памяти, либо пока не обнулится BC, т.о. эта инструкция может быть использована для поиска ячейки, содержащей определенное число.

Адресные регистры "DE" и "HL"

Регистры D и E не имеют особых функций и в основном используются в качестве временной быстро доступной памяти, они также используются в паре для хранения адреса ячейки, которая интересует нас в текущий момент.

Основная функция регистров H и L - хранить в паре адрес ячейки памяти. Мы уже видели, как определенная инструкция использует пару HL с этой целью. H хранит старший байт, L - младший. Полный адрес доступен в форме:

адрес = 256* H + L,

что дает максимум 65536 возможных адресов.

Индексные регистры "IX" и "IY"

Индексные регистры IX и IY являются 16-битовыми регистрами и могут быть использованы только так в отличие от регистров BC, DE и HL, которые могут использоваться и в паре, как 16-битовые регистры, и индивидуально, как 8-битовые. IX и IY, главным образом, используются подобно паре HL.

IX и IY, кроме того, имеют одно свойство, которое недоступно для HL - они могут быть использованы со смещением S. Имеется в виду, что инструкция, которая ссылается к (IX+S), использует не ячейку памяти, адрес которой хранится в IX, а сначала смещение S прибавляется к значению в IX, чтобы получить новый адрес, и с ним и работает инструкция.

Указатель стека "SP"

Стек - это область в верхней части ОЗУ, которая используется для временного хранения содержимого пар регистров. Стек растет вниз, когда идет заполнение и поднимается вверх при его опустошении. Начало стека фиксировано в ZX SPECTRUM - оно находится непосредственно ниже ячейки, на которую указывает системная переменная RAMTOP. Вершина стека находится ниже нижней границы стека, т.к. стек увеличивается вниз, а уменьшается вверх. Адрес текущей ячейки верхней границы стека хранится в SP регистре. Передача в стек или из стека происходит с помощью инструкций PUSH и POP. Например:

```
PUSH HL
```

В этом случае процессор:

1) Уменьшает SP;

2) Копирует содержимое регистра H в ячейку, указанную указателем стека SP;

3) Уменьшает SP;

4) Копирует содержимое регистра L в ячейку, указанную указателем стека SP.

Инструкция POP выполняет противоположное действие. По этому методу последняя поступившая пара чисел, помещенная в стек, является всегда первой парой, которая снимается со стека. Это обеспечивает простой и удобный способ временного хранения содержимого регистров во время вызова подпрограмм.

Программный счетчик "PC"

Программный счетчик PC очень важный 16-битовый регистр, т.к. он хранит адрес следующей инструкции, которая должна быть выполнена. Нормальный ход работы когда инструкция выполняется следующим образом:

1) Скопировать содержимое ячейки, указанной программным счетчиком PC в специальный регистр процессора.

2) Если инструкция содержится в нескольких байтах, увеличить PC и скопировать содержимое следующей ячейки во второй специальный регистр.

3) Увеличить PC таким образом, чтобы он указывал на следующую инструкцию, которая должна быть выполнена.

4) Выполнить инструкцию, которая только что была прочитана. Команда JUMP так же, как и DJNZ n или JR z,n , изменяет нормальный ход выполнения программы с помощью изменения PC во время выполнения шага 4. Заметим, что это изменение имеет место после увеличения PC. Таким образом, значение смещения n всегда будет отсчитываться относительно позиции инструкции, следующей за инструкцией, содержащей смещение.

Альтернативные регистры AF',BC',DE',HL'

Z80A имеет копию каждого из A, B, C, D, E, H и L регистров. Отличаются копии использованием знака " ' " возле обозначения регистра. Например, A' - это копия регистра A. Инструкции не работают с этими регистрами непосредственно, но инструкции обмена позволяют вывести из использования 2 или более регистров и ввести в использование вместо них их копии.

Команды обмена выполняются очень быстро, т.к. содержимое регистра физически не копируется из одного регистра в другой, а вместо этого установка внутренних коммуникаций процессора изменяется так, что дублирующий регистр становится основным, а оригинальный регистр запасным.

О системе команд процессора Z80

В системе команд Z80 около 700 инструкций. Поскольку возможны только 256 различных комбинаций из 8 битов (2^8=256), то лишь менее половины команд может быть выражено одним байтом. Оставшиеся инструкции хранятся в двух или трех байтах. Некоторые команды следуют с однобайтовым смещением S, или однобайтовым числом n, или двухбайтовым числом (адресом) nn, к которым инструкция ссылается. Всего же одна инструкция может занять максимум 4 байта.

* * *

ПРИМЕЧАНИЕ "ИНФОРКОМа"

ZX SPECTRUM располагает еще встроенным программируемым калькулятором, имеющим свою систему команд. Доступ к ним возможен только из машинного кода. Команды калькулятора могут быть более длинными, чем команды процессора и занимать более 4-х байт. (См. т. 1 нашего трехтомника).

* * *

Для удобства работы на ассемблере все команды записываются с помощью мнемоник (OP CODE). Мнемоника - это сокращенное описание каждой команды.

Правила задания команд в ассемблере:

1. Одиночные регистры описываются буквой, например - B. Пара регистров именуется в алфавитном порядке, например, BC.

2. Смещение S положительно, если оно находится в пределах от 0 до 127, и отрицательно, если оно находится в пределах от 128 до 255. Большие или меньшие числа недопустимы.

Отрицательное значение подсчитывается вычитанием S из 256. Например, команда относительного безусловного перехода:

```
JR S
```

вызывает переход вперед на 8 байтов, если S=8 и переход назад на 8 байтов, если S=248, т.к.

256-8=248.

Запомните при подсчете смещения, что переход осуществляется с адреса первого байта следующей команды.

3. Однобайтовое число n лежит в пределах от 0 до 255 включительно.

4. Двухбайтовое число или адрес представляется как nn и лежит в пределах от 0 до 65535 включительно.

5. nn, заключенное в скобки, т.е. (nn), подразумевает "содержимое ячейки по адресу nn", тогда как nn без скобок подразумевает "число nn".

Т.о.,

```
LD HL,(23627)
```

подразумевает загрузку пары регистров HL содержимым ячеек 23627 и 23628, тогда как:

```
LD HL,23627
```

подразумевает загрузку HL числом 23627.

Таким же образом, (HL) подразумевает "содержимое ячейки с адресом, хранящимся в

HL", тогда как HL без скобок означает "число, хранящееся в HL".

Такой вид адресации называется косвенной адресацией.

6. Место назначения результата операции всегда задается первым, например:

```
ADD A,B
```

означает "прибавить содержимое регистра B к содержимому регистра A и результат оставить в регистре A".

РАЗДЕЛ B

4. ВВЕДЕНИЕ

В Разделе B представлены программы в машинном коде. Для простоты и удобства использования они даны в стандартном формате. Во введении описывается этот формат и дается программа на BASICe, которая может быть использована для загрузки программ в память.

Длина: Это длина программы в байтах.

Количество: Выполнение некоторых программ может потребовать переменных: изменения значения одной или более переменных, передаваемых в программу через буфер принтера.

Контрольная сумма: Каждая программа дана в виде последовательности целых положительных чисел, помещаемых в последовательные ячейки памяти. Контрольная сумма (т. е. сумма всех чисел вводимой подпрограммы) дается для того, чтобы Вы были уверены в правильности загрузки

Функция: Дается краткое описание задачи, выполняемой с помощью программы.

Переменные: Определяются имя, длина и адрес в буфере принтера каждой переменной. Переменная длиной в один байт (целое положительное число в пределах от 0 до 255) передается в программу из BASICa или с клавиатуры через POKE.

POKE ячейка, значение

Двухбайтовая переменная передается с помощью 2-х команд:

```
POKE ячейка, значение - 256*INT(значение/256):
POKE ячейка+1,INT(значение/256)
```

Используемые ячейки являются ячейками памяти буфера принтера

Вызов подпрограммы: Программы вызываются с использованием функции USR, которая должна быть включена в команду. Если программа в машинном коде не передает какое-то значение обратно в BASIC по завершению, то используется команда:

```
RANDOMIZE USR адрес
```

Если Вам надо, чтобы результат был возвращен в регистровой паре BC, то вызов делается так:

```
LET A = USR адрес
```

или

```
PRINT USR адрес
```

в зависимости от того, должны ли возвращаемые данные сохраняться в переменной BASICa или выводиться на экран.

Контроль ошибок:	Объясняются проверки, выполняемые программой для нелогичных или противоречивых значений переменных, параметров и т. п.
Комментарии:	Объясняются возможные варианты в программах.
Листинг в машинном коде:	Программы представлены на языке ассемблера. Для загрузки в память используется третья колонка - "числа для ввода". Все числа здесь даны в десятеричной системе.
Как она работает:	Объяснение принципа работы программы.

Загрузчик машинного кода

Почти все программы из этой книги перемещаемые, т.е. они будут работать корректно независимо от того, в каком месте RAM мы их поместим. Если программа не перемещаемая, то в комментариях объясняется, как она должна быть изменена, если ее нужно сохранить в другой области памяти. В Разделе А (часть 2) мы видели, что SPECTRUM использует различные части RAM для различных функций и что область между ячейками, указанными системными переменными RAMTOP и UDG предназначена для хранения подпрограмм в машинном коде.

Программа BP может быть использована для загрузки, изменения и перемещения программы в машинном коде. С ее помощью пользователь может переустановить указатель RAMTOP, что даст больше свободного пространства для машинных кодов; ввести программу с клавиатуры; перейти вперед или назад для корректировки ошибки; вставить или удалить часть программы.

Когда программа BP запускается, она печатает младший адрес, с которого программа в машинных кодах может быть введена и сохранена, т.е. на единицу больший, чем RAMTOP.

В машине с 48K памяти младший адрес - 65368, если пользователь не обновлял системную переменную RAMTOP. В конце ОЗУ обычно резервируются 168 байтов для UDG, но программа позволяет пользователю использовать и эту область, если он пожелает. Он может также выбрать новый возможный младший адрес, который программа затем помещает в системную переменную RAMTOP, используя команду CLEAR. Данные не могут быть введены по адресу, меньшему, чем 27000, т.к. иначе нарушатся границы области, требующейся для самой программы BP. Программа BP запрашивает адрес, с которого должна стартовать программа в машинных кодах. Т.о. пользователь может резервировать область для нескольких процедур и затем загружать их каждую отдельно.

Рисунок BF1 показывает формат дисплея после того, как с ячейки 32000 была загружена программа "screen invert". Первая колонка - адрес, вторая - содержимое ячейки памяти с этим адресом, третья - контрольная сумма. Программа "screen invert" - имеет 18 байтов в длину и ее контрольная сумма = 1613. Следовательно, она занимает ячейки от 32000 до 32017, и контрольная сумма дана для ячейки 32017, т. е. сумма содержимого ячеек (32000...32017) равна 1613.

На основном экране внимание пользователя привлекается к одной ячейке - содержимое этой ячейки мерцает. Эта ячейка является текущей и первоначально это выбранный стартовый адрес программы. Пользователь вводит целое число между 0 и 255 включительно, которое программа MC LOADER помещает в текущую ячейку. Затем следующий адрес становится текущей ячейкой. Пользователь может не вводить число, а вместо этого выбрать для корректировки вариант, описанный в таблице BT1.

Код	Вариант
B	Перейти на один адрес назад.
B n(число)	Перейти на n адресов назад.
F	Перейти на один адрес вперед.
F n(число)	Перейти на n адресов вперед.
I n(число)	Вставить n байтов, каждый из которых содержит 0.
D n(число)	Удалить n байтов в текущей области.
T	Закончить программу

Таблица BT1. Возможные варианты редактирования машинного кода

Программа BP. загрузчик машинного кода. (MC LOADER)

```
100 GO SUB 8100
200 REM ***** Вычисление доступной памяти
210 LET min= 1+PEEK 23730+256*PEEK 23731
220 LET P = PEEK 23732+ 256*PEEK 23733
230 LET t = P - min + 1
400 REM ***** Определение стартового адреса
410 PRINT "Lowest possible start - ";min,,,"Maximum space availbie = ";t
420 INPUT "Do you wish to change the lowest start address (Y or N) ?";z$
430 IF Z$="Y" OR Z$="y" THEN GO TO 7000
440 INPUT "Enter address at which to start loading machine code";a
450 IF a<min OR a>p THEN BEEP .2,24: GO TO 440
500 GO SUB 8100
510 LET t=t-a+min
520 PRINT "You can use up to ";t;" bytes",,,
530 LET U=PEEK 23675+256*PEEK 23676
540 IF a<u AND u<p THEN PRINT "If you use more than ";u-a;" bytes, you will overwrite
      the user defined graphics area. "
550 IF a>=U THEN PRINT "You will overwrite the user defined graphics area."
560 INPUT "Is that OK (Y or N) ? ";z$
570 IF Z$="N" OR z$="n" THEN GO TO 7000
580 IF Z$<>"Y" AND z$<>"y" THEN BEEP .2,24: GO TO 560
700 REM *** GO AHEAD AND LOAD
710 LET i=a
750 GO SUB 8200
760 INPUT "Enter number, b, f, i, d or t "; z$
770 IF z$="" THEN BEEP .2,24: GO TO 760
780 LET a$=CHR$(CODE Z$(i) - 32*(Z$(i)>"@"))
790 GO TO 800+200*(a$="B")+300*(a$="F")+400*(a$="I")+500*(a$="D")+600*(a$="T")
800 LET X=VAL Z$
810 IF i>p THEN BEEP .2,24: GO TO 750
820 IF X<0 OR X>256 OR X<>INT X THEN BEEP .2,24: GO TO 760
830 POKE i,X
840 LET i=i+1
850 GO TO 740
1000 REM *** Перемещение вперед
1010 LET i=i-1
1020 IF LEN Z$>1 THEN LET i=i+1-VAL Z$(2 TO)
1030 IF i<a THEN LET i=a
1040 GO TO 740
1100 REM *** Перемещение назад
1110 LET i=i+1
1120 IF LEN Z$>1 THEN LET i=i-1+VAL Z$(2 TO)
1130 IF i>p THEN LET i=p
1140 GO TO 740
1200 REM *** Вставка
1210 IF LEN Z$=1 THEN LET n=1: GO TO 1225
1220 LET n = VAL Z$(2 TO): IF n<1 OR n>p-1 OR n<>INT n THEN BEEP .2,24: GO TO 740
1225 CLS: GO SUB 8100: PRINT TAB 6; "Inserting in Progress"
1230 FOR J=p TO i+n STEP -1
1240 POKE J,PEEK (j-n)
1250 NEXT J
1260 FOR J=1 TO i+n-1
1270 POKE J,0
1280 NEXT J
1290 GO TO 740
1300 REM *** Удаление
1310 IF LEN z$=1 THEN LET n=1: GO TO 1330
1320 LET n=VAL z$(2 TO): IF n<l OR n>P-1 OR n<>INT n THEN BEEP .2,24: GO TO 740
1330 IF n<0 OR n>p-1 THEN BEEP .2,24: GO TO 1320
1340 CLS: GO SUB 8100 :PRINT TAB 6; "DELETING IN PROGRESS"
1350 FOR J=1 TO p-n
```

```
1360 POKE J,PEEK (j+n)
1370 NEXT J
1380 GO TO 740
1400 STOP
1401 PRINT AT 21,7;"PROGRAM TERMINATED"
1410 STOP
7000 REM *** Изменение RAMTOP
7010 INPUT "ENTER NEW START ADDRESS ";a
7020 IF a<27000 OR a>p THEN BEEP .2,24: GO TO 710
7030 CLEAR a-1
7040 RUN
7999 STOP
8100 CLS
8110 PRINT TAB 6; "MACHINE CODE LOADER",,,
8120 RETURN
8200 REM *** Вывод содержимого памяти
8210 GO SUB 8100
8220 PRINT "ADDRESS DECIMAL CHECK SUM"
8230 LET c=0
8240 LET s=i-8 : IF s<a THEN LET s=a : GO TO 8280
8250 FOR j=a TO s-i 8260 LET c=c+PEEK j
8270 NEXT j
8280 LET f=s+17 : IF f>p THEN LET f=p
8290 FOR j=s TO f
8300 LET c=c+PEEK j
8310 PRINT AT j-s+3,i;j: TAB 12:PEEK j; TAB 22;c
8320 NEXT j
8400 LET pos=i-s+3
8410 PRINT AT pos,12; FLASH 1; PEEK i
8420 RETURN
```

MACHINE CODE LOADER

ADDRESS	DECIMAL	CHECK SUM
32000	33	33
32001	0	33
32002	64	97
32003	1	98
32004	0	98
32005	24	122
32006	22	144
32007	255	399
32008	122	521
32009	150	671
32010	119	790
32011	35	825
32012	11	836
32013	120	956
32014	177	1133
32015	32	1165
32016	247	1412
32017	201	1613

Рис. ВР1.

Представлен экран во время работы загрузчика машинного кода - программа "Screen Invert" загружена с адреса 32000.

5. Подпрограммы сдвига.

5.1 Сдвиг атрибутов влево.

Длина: 23

Количество переменных: 1

Контрольная сумма: 1574

Назначение: эта программа сдвигает атрибуты всех символов экрана влево на одно знакоместо.

Переменные:

Имя: new attr

Длина: 1

Ячейка: 23296

Комментарий: это атрибут, вводимый в крайнюю правую колонку.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок:

Нет

Комментарий: Эта программа полезна для выделения области текста или графики. Для прокручивания только 22 верхних строк 24* должно быть изменено на 22.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,22528        33    0     88
            LD A,(23296)       58    0     91
            LD C,24            14    24*
NEXT_L      LD B,31            6     31
NEXT_C      INC HL             35
            LD E,(HL)          94
            DEC HL             43
            LD (HL),E          115
            INC HL             35
            DJNZ NEXT_C        16    249
            LD (HL),A          119
            INC HL             35
            DEC C              13
            JR NZ,NEXT_L       32    242
```

Как она работает:

В пару регистров HL загружается адрес области атрибутов. Аккумулятор загружается значением атрибута, вводимым в правую колонку. В регистр C загружается количество строк для сдвига - он теперь может быть использован, как счетчик строк. В регистр B заносится число на 1 меньшее, чем число символов в строке, чтобы он использовался, как счетчик. HL увеличивается, чтобы указать на следующий атрибут, который загружается в E-регистр. HL уменьшается и по адресу HL помещается значение из E-регистра. HL увеличивается вновь, чтобы указать на следующий атрибут. Регистр B уменьшается и, если он не равен 0, то происходит переход к NEXT_C'. HL теперь указывает на правую колонку и по адресу HL помещается значение из аккумулятора. HL увеличивается, чтобы указать на следующую строку - NEXT_L. Счетчик строк (регистр C) уменьшается. Если значение результата не равно 0, подпрограмма возвращается назад к NEXT_L.

По окончании работы программа возвращается в BASIC.

5.2 Сдвиг атрибутов вправо.

Длина: 23

Количество переменных: 1

Контрольная сумма: 1847

Назначение: Эта программа сдвигает атрибуты всех символов экрана вправо на одно знакоместо.

Переменные:

Имя: new attr

Длина: 1

Ячейка: 23296

Комментарий: это атрибут, вводимый в крайнюю левую колонку.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: нет

Комментарий: Эта программа полезна для выделения области текста или графики. Для прокручивания только 22 верхних строк 24* должно быть изменено на 22.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,23295        33    255   88
            LD A,(23296)       58    0     91
            LD C,24            14    24*
NEXT_L      LD B,31            6     31
NEXT_C      DEC HL             43
            LD E,(HL)          94
            INC HL             35
            LD (HL),E          115
            DEC HL             43
            DJNZ,NEXT_C        16    249
            LD (HL),A          119
            DEC HL             43
            DEC C              13
            JR NZ,NEXT_L       32    242
            RET                201
```

Как она работает:

В пару регистров HL загружается адрес последнего байта области атрибутов. Аккумулятор загружается значением атрибута для ввода в левую колонку. В регистр C загружается количество строк для сдвига - он используется, как счетчик строк. В регистр B заносится число на 1 меньшее, чем число символов в строке для использования его в качестве счетчика.

HL увеличивается, чтобы указать на следующий атрибут. Значение этого атрибута загружается в E-регистр. HL увеличивается, и по адресу HL помещается значение из E-регистра. HL уменьшается снова для указания адреса следующего атрибута. Счетчик в регистре B уменьшается и, если он не равен 0, то - возврат назад к NEXT_C. HL теперь указывает на крайнюю левую колонку, и в ячейку с адресом HL, помещается значение из аккумулятора. HL уменьшается для указания правого конца следующей строки. Счетчик строк уменьшается, и если он не равен 0, - возврат назад к NEXT_L.

Программа возвращается в BASIC.

5.3 Сдвиг атрибутов вверх.

Длина: 21

Количество переменных: 1

Контрольная сумма: 1591

Назначение: Эта программа сдвигает атрибуты всех символов экрана вверх на одно знакоместо.

Переменные:

Имя: new attr

Длина: 1

Ячейка: 23296

Комментарий: это атрибут, вводимый в нижнюю строку.

Вызов программы:

```
RAMDOMIZE USR адрес
```

Контроль ошибок: нет

Комментарий: эта программа полезна для выделения области текста или графики. Для прокручивания только 22 верхних строк 224* должно быть изменено на 160.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,22560        33    32    88
            LD DE,22526        17    0     88
            LD BC,736          1     224*  2
            LDIR               237   176
            LD A,(23296)       58 0  91
            LD B,32 6          32
NEXT_C      LD (DE),A          18
            INC DE             19
            DJNZ NEXT_C        16    252
            RET 201
```

Как она работает

В пару регистров HL загружается адрес второй строки атрибутов, в DE загружается адрес первой строки, и BC загружается числом байтов для перемещения.

Количество байтов, определенное в регистровой паре BC, копируются в ячейки памяти с адресом, находящимся в DE из ячеек, начинающихся с адреса в HL, используя инструкцию LDIR. Эти результаты в DE указывают на нижнюю строку атрибутов. Аккумулятор загружается кодом атрибута для ввода в нижнюю строку. B-регистр загружается числом символов на одной строке - он используется, как счетчик.

По адресу DE помещается значение из аккумулятора, а затем DE увеличивается для указания на следующий байт. Счетчик уменьшается и, если он не равен 0, подпрограмма возвращается к NEXT_C.

Затем программа возвращается в BASIC.

5.4 Сдвиг атрибутов вниз.

Длина: 21

Количество переменных: 1

Контрольная сумма: 2057

Назначение: эта программа сдвигает атрибуты всех символов экрана вниз на одно знакоместо.

Переменные:

Имя: new attr

Длина: 1

Ячейка: 23296

Комментарии: это атрибут, вводимый в верхнюю строку.

Вызов программы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: нет

Комментарий: эта программа полезна для выделения области текста или графики, для прокручивания только 22 верхних строк должны быть изменены: 223* на 159 255* на 191 224* на 160

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,23263        33    223   90
```

```
            LD DE,23295         17     255    90
            LD BC,736           1      224    2
            LDDR                237    184
            LD A,(23296)        58     0      91
            LD B,32             6      32
NEXT_C      LD (DE),A           18
            DEC DE              27
            DJNZ NEXT_C         16     252
            RET                 201
```

Как она работает:

В HL загружается адрес последнего атрибута 23-й строки. В DE загружается адрес последнего атрибута 24-й строки. В BC загружается число байтов для перемещения, затем команда LDDR перемещает байты (их количество указано в регистровой паре BC) из адреса в HL по адресу в DE. Эти результаты в DE хранят адрес последнего атрибута первой строки.

В аккумулятор загружается значение атрибута для ввода в верхнюю строку. В B регистр загружается число байтов в верхней строке - он используется, как счетчик. В ячейку с адресом DE помещается значение из аккумулятора и DE уменьшается для указания на следующий байт. Счетчик уменьшается и, если он не равен 0, подпрограмма возвращается к NEXT_C. Программа затем возвращается в BASIC.

5.5 Сдвиг влево на один символ.

Длина:21

Количество переменных:0

Контрольная сумма:1745

Назначение: эта программа прокручивает графику экрана на один символ влево. Вызов подпрограммы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий:

Эта программа полезна при организации экрана в качестве "окна", показывающего в данный момент меньшую часть большой дисплейной области. Это "окно" перемещается, используя подпрограммы сдвига.

ЛИСТИНГ МАШИННЫХ КОДОВ

```
МЕТКА       АССЕМБЛЕР           ЧИСЛА ДЛЯ ВВОДА

            LD HL,16384         33     0      54
            LD D,L              85
            LD A,192            62     192
NEXT_L      LD B,31             6       31
NEXT_B      INC HL              35
            LD E,(HL)           94
            DEC HL              43
            LD (HL),E           115
            INC HL              35
            DJNZ NEXT_B         16     249
            LD (HL),D           114
            INC HL              35
            DEC A               61
            JR NZ,NEXT_L        32     242
            RET                 201
```

Как она работает:

В пару регистров HL загружается адрес дисплейного файла, а D-регистр устанавливается в 0. В аккумулятор загружается число строк на экране. В B-регистр загружается значение на 1 меньшее, чем число символов в строке - оно является числом байтов для копирования.

HL увеличивается для указания на следующий адрес и содержимое из этой ячейки загружается в E-регистр. HL уменьшается и в ячейку с адресом HL загружается значение из E-регистра. HL увеличивается для адресации следующего байта и счетчик в B-регистре уменьшается. Если он не равен 0, подпрограмма возвращается к NEXT_B.

Если регистр B равен 0, это означает, что последний байт строки скопирован и HL указывает на крайний правый байт. По этому адресу в регистровой паре HL помещается 0 и HL увеличивается, указывая на следующую строку. Счетчик строк - аккумулятор - уменьшается, и, если он не равен нулю, происходит переход к NEXT_L.

Программа возвращается в BASIC.

5.6 Сдвиг вправо на один символ.

Длина:22

Количество переменных:0

Контрольная сумма:1976

Назначение: эта программа прокручивает содержимое дисплейного файла на один символ вправо.

Вызов подпрограммы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: нет

Комментарий:

Эта программа полезна при организации экрана в качестве "окна", показывающего в данный момент меньшую часть большой дисплейной области. Это "окно" перемещается, используя подпрограммы сдвига.

Листинг машинных кодов

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,22527        33    255   87
            LD D,0             22    0
            LD A,192           62    192
NEXT_L      LD B,31 6          31
NEXT_B      DEC HL             43
            LD R,(HL)          94
            INC HL             35
            LD (HL),E          115
            DEC HL             43
            DJNZ NEXT_B        16    249
            LD (HL),D          114
            DEC HL             35
            DEC A              61
            JR NZ,NEXT_L       32    242
            RET                201
```

Как она работает:

В пару регистров HL загружается адрес последнего байта дисплейного файла, а D-регистр устанавливается в 0. В аккумулятор загружается число строк на экране. В B-регистр загружается значение на 1 меньшее, чем число символов в строке, - он используется, как счетчик.

HL уменьшается, указывая на следующий байт, и его значение загружается в E-регистр. HL затем увеличивается, и в ячейку с адресом HL загружается значение E-регистра. HL уменьшается, указывая на следующий байт, и счетчик (B-регистр) уменьшается. Если он не равен 0, подпрограмма возвращается к NEXT_B.

Если регистр B равен 0, это означает, что HL указывает на крайний левый байт строки. Затем по адресу в регистровой паре HL помещается 0, и HL уменьшается, указывая на следующую строку. Счетчик строк (аккумулятор) уменьшается и, если он не равен 0, происходит переход к NEXT_L.

Программа возвращается в BASIC

5.7 Сдвиг вверх на один символ

Длина: 68

Количество переменных:0

Контрольная сумма: 6326

Назначение: программа сдвигает содержимое дисплейного файла вверх на восемь пикселей.

Вызов подпрограммы:

```
RANDOMIZE USR адрес
```

Контроль ошибок: Нет

Комментарий: нет

Листинг машинных кодов

```
МЕТКА       АССЕМБЛЕР          ЧИСЛА ДЛЯ ВВОДА

            LD HL,16364        33    0     64
            LD DE,16416        17    32    64
SAVE        PUSH HL            229
            PUSH DE            213
            LD C,23            14    23
NEXT_L      LD B,32            6     32
COPY_B      LD A,(DE)          26
            LD (HL),A          119
            LD A,C             121
            AND 7              230   7
            CP 1               254   1
            JR NZ,NEXT_B       32    2
            SUB A              151
            LD (DE),A          18
NEXT_B      INC HL             35
            INC DE             19
            DJNZ COPY_B        16    241
            DEC C              13
            JR Z,REST          40    19
            LD A,C             121
            AND 7              230   7
            CP 0               254   0
            JR Z,N_BLOCK       40    22
            CP 7               254   7
            JR NZ,NEXT_L       32    225
            PUSH DE            213
            LD DE,1792         17    0     7
            ADD HL,DE          25
            POP DE             209
            JR NEXT_L          24    217
REST        POP DE             209
            POP HL             225
            INC D              20
            INC H              36
            LD A,H             124
            CP 72              254   72
            JR NZ,SAVE         32    204
            RET                201
N-BLOCK     PUSH HL            229
            LD HL,1792         33    0     7
            ADD HL,DE          25
            EX DE,HL           235
            POP HL             225
            JR NEXT_L          24    198
```

Как она работает:

В пару регистров HL загружается адрес начала дисплейного файла, а в DE загружается адрес байта через восемь линий вниз. HL и DE сохраняются в стеке, в C-регистр загружается число на 1 меньшее, чем число строк на экране. В B-регистр загружается количество байтов на одной линии дисплея - он используется, как счетчик.

В аккумулятор загружается байт, адресованный DE, и это значение загружается в ячейку по адресу HL. В аккумулятор загружается содержимое C-регистра и, если оно равно 1, 9 или 17, то в ячейку по адресу DE помещается 0. HL и DE увеличиваются, указывая на следующий байт, счетчик в B-регистре уменьшается и, если он не равен 0, происходит переход к 'COPY_B'.

Счетчик строк в регистре C затем уменьшается. Если он равен 0, происходит переход к 'RESTORE'. Если C содержит 8 или 16, то происходит переход к 'N_BLOCK'. Если C не содержит 7 или 15, подпрограмма переходит к 'NEXT_L'. Затем 1792 прибавляется к HL - теперь HL указывает на следующий блок экрана. Подпрограмма переходит к 'NEXT_L'.

В процедуре 'REST' DE и HL берутся из стека и 256 прибавляется к каждому из них. Т.о., DE и HL указывают на строку, позиция которой ниже, чем та, что была в предыдущем цикле. Если HL содержит значение 18432, подпрограмма возвращается в BASIC, иначе происходит переход к процедуре 'SAVE'. В процедуре 'N_BLOCK' 1792 прибавляется к DE - таким образом DE указывает на следующий блок экрана. Подпрограмма затем возвращается к 'NEXT_L'.

По окончании работы происходит возврат в БЕЙСИК.

Продолжение следует

Алексеев А. Г.

МАСТЕРФАЙЛ 09 полная русификация.

ВСТУПЛЕНИЕ.

Прежде всего я хотел бы сказать что эта статья ориентирована на тех пользователей "Спектрума" которые уже немного овладели программированием на Бейсике, зашитом в ПЗУ компьютера но, еще не владеют знаниями, необходимыми для программирования в машинных кодах Z80. Но, пользуясь только этими знаниями, многие усовершенствования можно уже делать с готовыми программами. А стимул для дальнейшего освоения машинных кодов несомненно появится в процессе такой работы. Таким образом, психологический барьер будет преодолён и Вы выйдете на новый уровень.

В этой статье изложены основные методы и приемы неполной и полной русификации программ для "Спектрума". В качестве примера будут рассмотрены способы русификации программы "MF 09". Она достаточно широко распространена среди пользователей "Спектрума" и английский вариант сильно сдерживает ее применение.

Существует несколько способов русификации "Спектрума". Сначала немного о самом простом из них.

1. Использование символов UDG-графики.

Так как многие латинские буквы в режиме CAPS LOCK по написанию совпадают с русскими (A, B, E, K, M, H, O, P, C, T, X), а вместо русской буквы "З" можно использовать цифру "3", то остается всего 20 букв, требующихся для русификации (Б, Г, Д, Ж, И, Й, А, П, У, Ф, Ц, Ч, Ш, Щ, Ъ, Ы, Ь, Э, Ю, Я), то есть мы укладываемся в 21 символ, которые отведены для графики пользователя в "Спектруме".

Предлагаемая программа наглядно показывает принцип формирования символа UDG-графики. Например, для формирования буквы "Я", закрепленной за символом UDG "A", набираем программу:

```
10 LET n=USR "a"
20 FOR x=n TO n+7
30 READ y
40 POKE x,y
50 NEXT  x
100 DATA
      BIN 00000000,
      BIN 00111110,
      BIN 01000010,
      BIN 01000010,
      BIN 00111110,
      BIN 00100010,
      BIN 01000010,
      BIN 00000000
```

В строке 100 после запятых надо набирать по 19 пробелов, чтобы наглядно просматривалась будущая буква. Те пиксели, которые должны быть включены, отмечаем как "1", а те, которые выключены - как "0".

Для буквы "Б" надо в строке 10 вместо USR "a" подставить USR "b" и изменить нули и единицы в строке 100.

Более подробно этот метод был изложен в разработке ИНФОРКОМА "Большие возможности Вашего Спектрума". Говорилось о нем и в ZX-РЕВЮ N 4-5 (стр. 80). Поэтому,

чтобы не повторяться, представим теперь, что русские UDG-символы Вами уже сформированы и Вы записываете 21 символ, начиная с "a", на магнитофон:

```
SAVE "rusudg" CODE USR "a",21*8
```

При наборе своих собственных программ можно поступать, например, следующим образом. После рестарта компьютера набрать:

```
1 GO TO 100
2 LOAD "rusudg"  CODE
3 GO TO 1
5 SAVE  "zagotowka" LINE 2:
SAVE "rusudg"  CODE USR "a",168
6 GO TO 5
```

Получился своеобразный "дебют" программы. Теперь сделайте RUN 2 и загрузите записанную ранее область UDG. После сообщения "0 OK" набирайте свою программу, начиная с той строки, которую указали в строке 1 после GO TO, то есть с 100. Для записи готовой программы на магнитофон используйте RUN 5, предварительно подставив вместо "zagotowka" имя программы. При этом запишется программа, а следом за ней блок кодов UDG. Строка 6 "зацикливает" процесс записи, если Вам надо записать несколько дублей программы. При загрузке программа автостартует со строки 2 и загрузит блок UDG. Команда RUN обеспечивает "холодный" старт программы.

Блок кодов UDG-графики, также, как и любой другой блок кодов, можно разместить внутри Бейсик-программы, используя для этого нулевую строку. Это имеет определенные преимущества: программа состоит не из двух, а из одного куска, сокращается время загрузки. Делается это очень просто.

После рестарта компьютера наберите: 1 REM, а после REM столько пробелов (или любых других символов), сколько байтов памяти Вам надо зарезервировать для Ваших целей. Для размещения, например, блока символов UDG надо набрать 168 пробелов. После того, как строка введена в память компьютера, заменяем ее номер на 0, подав прямую команду: POKE 23756,0 . На экране теперь видим: 0 REM. Первая строка стала нулевой. Теперь ее невозможно случайно испортить, вызвав на редактирование командой EDIT. Свободная область в этой строке начинается с первого символа (пробела), стоящего за REM, адрес ее начала - 23760, длина равна числу набранных символов (пробелов). Теперь, если Вы набрали 168 пробелов, можете загрузить в нулевую строку блок кодов UDG-графики:

```
LOAD "rusudg" CODE 23760
```

Далее надо сделать так, чтобы включался блок UDG-кодов, загруженный в новое место. Для этого используем системную переменную UDG, занимающую два байта в ячейках 23675 и 23676. Эта переменная указывает адрес первой ячейки области UDG. Выполните:

```
PRINT PEEK 23675+256*PEEK 23676
```

Вы получите результат: 65368. Кстати, этот же результат Вы получите, выполнив: PRINT USR "a".

Новое значение системной переменной UDG будет равно 23760. Вычислим младший, а затем старший байты этого числа, выполнив:

```
PRINT 23760-256*INT(23760/256)
PRINT INT(23760/256)
```

Получим 208 и 92. Теперь догрузим к нулевой строке наш "дебют" программы, выполнив:

```
MERGE "zagotowka"
```

и изменим строки 2 и 5:

```
1 GO TO 100
2 POKE 23675,208: POKE 23676,92
3 GO TO 1
5 SAVE "zasotowka" LINE 2
```

```
6 GO TO 5
```

Теперь сделайте RUN 2. После сообщения "0 OK" можете набирать свою программу, используя графический регистр. Надо только помнить о том, что при выполнении команды LIST при выведении нулевой строки, скорее всего, будет сообщение об ошибке. Это связано с тем, что интерпретатор Бейсика не понимает ту информацию, которая стоит в нулевой строке. Для получения листинга придется делать LIST 1. Неприятности могут быть также и в том, что не все начальные строки могут быть видны в автоматическом листинге, хотя они нормально вызываются при редактировании командой EDIT и работа готовой программы от этого никак не страдает.

Для несложных своих программ этот способ русификации вполне может применяться, так как позволит писать на русском языке даже комментарии в тексте программы.

Что касается программы MF 09, то, к сожалению, к ней не подходит этот самый простой способ русификации, так как символы, набранные с использованием графического регистра (курсор [G]) не отображаются в режиме "дисплей". Вместо них на экране печатается знак "?". Такие же трудности возникнут и при выведении текста на печать при помощи принтеров, отличных от "ZX". Поэтому рассмотрим другой, более совершенный способ неполной русификации.

2. Использование дополнительного символьного набора.

Некоторые проблемы, связанные с этим вопросом, были изложены в ZX-РЕВЮ № 4-5 стр. 80-81. Наиболее крупная из всех - это, по моему, программная совместимость. Какой латинской букве будет соответствовать какая русская? Это не имеет большого значения для конкретной игровой программы, где не надо вводить текст в процессе игры, однако для таких программ, как MF 09, это имеет немаловажное значение, ведь со временем все больше и больше появится возможностей пользоваться базами данных, составленными другими авторами. Это могут быть и каталоги программ для "Спектрума" (с комментариями на русском языке), и расписание движения поездов, самолетов и т.д. Даже если Вы вместе с базой данных перепишете и саму программу MF 09 автора, трудности возникнут при внесении дополнений и изменений в базу данных, так как каждый раз надо будет "осваивать" незнакомое распределение русских букв между кнопками клавиатуры.

К стандарту кодов ASCII автор этих строк подошел своим, независимым от ИНФОРКОМА путем, но результат все равно тот же. Значит наверняка есть и другие авторы, работающие в этом стандарте. Кстати и символьный набор "Спектрума" (коды с 32 по 127) являются кодами ASCII. Поэтому настоятельно рекомендую начинающим придерживаться именно этого стандарта. Распределение русских букв на первый взгляд может показаться неудобным, но Вы быстро привыкнете и перестанете это неудобство замечать. Зато у Вас будет больше шансов найти "привычные" для себя программы.

Базисная таблица КОИ-7 кодов ASCII имеет несколько символьных наборов: латинский, русский, смешанный. Символьный набор условно можно разделить на три части:

1 часть - коды с 32 по 63 - символы и цифры;

2 часть - коды с 64 по 95 - буквы, в основном печатаемые в регистре CAPS LOCK;

3 часть - коды с 96 по 127 - буквы, печатаемые без регистра CAPS LOCK.

Вы можете увидеть эти три части в трех строках на экране, выполнив:

```
FOR a=32 TO 127: PRINT CHR$ a;: NEXT a
```

То, что Вы видите на экране, является символьным набором "H0" базисной кодовой таблицы КОИ-7 кодов ASCII. Если взять другой символьный набор: "H1", то у него первая часть остается той же, вторая часть - это строчные русские буквы, а третья часть - заглавные русские буквы. Таблица их соответствия с тем набором, который в ROM "Спектрума" приведена в ZX-РЕВЮ №4-5 за 1991г., стр. 81.

Когда Вы принципиально решили для себя вопрос стандартизации, все остальное - дело техники. Загружаем программу "ART STUDIO", входим в режим TEXT и далее FONT EDITOR. Теперь надо переделать латинские буквы на русские соответственно таблице (вторую и третью части символьного набора), оставив без изменения цифры и символы (первую часть символьного набора) и записать новый набор символов на ленту (например с

именем "ruschr").

Полученный символьный набор занимает 768 байтов в памяти. Для использования в своих программах, его удобно расположить непосредственно перед символами UDG, разместив с адреса:

```
USR "a"-768=64600
```

Кстати, сохранять на ленте целесообразно символьный набор вместе с блоком кодов UDG; последние могут использоваться в Ваших программах непосредственно по назначению - для получения графических элементов. Для этого надо обьединить эти блоки в один:

```
LOAD "ruschr" CODE 64600,768
LOAD "rusudg" CODE 65368,168
SAVE "rus" CODE 64600,936
```

Теперь подумаем об удобстве пользования. Если символьный набор расположен с адреса 64600, то системная переменная CHARS (ячейки 23606, 23607) будет равна:

64600-256=64344

Младший и старший байты ее:

64344-256*INT(64344/255)=88

INT(64344/256)=251

"Дебют" программы может выглядеть теперь следующим образом:

```
1 GO TO 100
2 LOAD ""
3 GO TO 1
5 SAVE "zagotowka" LINE 2 : SAVE "rus" CODE 64600,936
6 GO TO  5
8 POKE 23606,88: POKE 23607, 251: RETURN : REM rus
9 POKE 23606,0 : POKE 23607,60 :RETURN : REM lat
```

Для записи готовой программы, как и раньше, используется RUN 5 (предварительно подставив имя программы). Следом за программой будет записан символьный набор с блоком UDG (т.е. 768+168=936 байт). Теперь поговорим о строках 8 и 9. Подайте прямые команды:

```
GO SUB 8 и затем: LIST
```

Вы увидите, что листинг программы печатается русскими буквами. Для обратного переключения сделайте GO SUB 9. Теперь эти переключения можно применять в Вашей программе. Например, чтобы написать фразу: "ZX-Spectrum пишет ПО-РУССКИ", надо ввести такую строку в программе:

```
100 PRINT "ZX-Spectrum ";:GO SUB 8: PRINT "pi{et PO-RUSSKI":GO SUB 9
```

Чтобы не допускать ошибок при наборе большого объема русского текста в своей программе, перед набором строки сделайте GO SUB 8 прямой командой и Вы будете видеть русский текст, который набиваете, в том виде, как он будет напечатан на экране.

Применение двух символьных наборов позволяет выводить на экран текст русскими и латинскими заглавными и строчными буквами, однако производить переключения вставляя в программу GO SUB 8 и GO SUB 9 не очень удобно, но главное, эти переключения нельзя выполнить, не останавливая программу, например, набирая строку данных в программе MF 09. Поэтому можно воспользоваться другим набором кодов ASCII - это набор КОИ-7 "НС". Здесь первая и вторая части - совпадают с ROM "Спектрума", а в третьей части вместо строчных латинских - заглавные русские буквы.

Таблица соответствия с ROM "Спектрума" приведена ниже (только третья часть символьного набора):

99	c	Ц	115	s	С
100	d	Д	116	t	Т
101	e	Е	117	u	У
102	f	Ф	118	v	Ж
103	g	Г	119	w	В
104	h	Х	120	x	Ь

105	i	И	121	y	И
106	j	Й	122	z	З
107	k	К	123	[	Ш
108	l	Л	184	верт. черта	Э
109	m	М	125	]	Щ
110	n	Н	126	апостроф	Ч
111	o	О	127	копирайт	Ъ

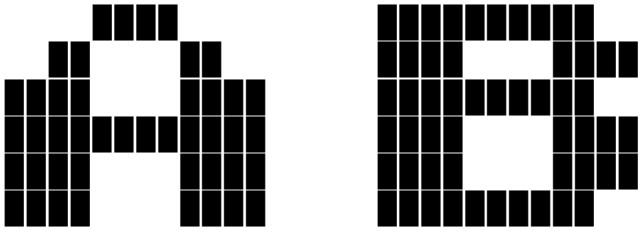
В этом случае переключение на символьный набор можно сделать только один раз сразу же после старта программы. Измените строку 3 нашего дебюта:

```
3 GO SUB 8: GO TO 1
```

При этом, правда, придется пользоваться хотя и русским и латинскими но только заглавными буквами.

Попутно следует сказать несколько слов о том, как выглядит готовый шрифт на экране компьютера. Вы обращали внимание, что почти ни в одной хорошей коммерческой программе не применяется символьный набор "Спектрума" а, в основном стилизованные шрифты. К особому зрительному эффекту приводит применение уплотнённых шрифтов. При этом программа буквально преображается. Почему бы не использовать такие шрифты в своих программах? Кроме красивого "внешнего вида" они имеют повышенную четкость при различном сочетании цветов INK - PAPER.

В своих программах я применяю "утолщенный" русско-латинский символьный набор ("НС" в кодах КОИ-7), который хочу предложить вашему вниманию. При этом буквы выглядят в увеличенном виде следующим образом (██ - один пиксел):



Однако большие массивы текста, состоящего только из заглавных букв, да еще "утолщенных", на экране выглядят слишком плотно и тяжело. Поэтому русские буквы в этом символьном наборе сделаны пониже, чем латинские. По высоте - как строчные буквы, а по написанию - как заглавные. При этом как бы увеличивается расстояние между строчками и текст выглядит более гармонично. Все то, о чем говорится в этих строках, может быть подвергнуто сомнению. К тому же набирать 768 чисел вручную утомительно. Но попробуйте, хотя бы из любопытства я уверен: то, что Вы получите, вам понравится. Одни явные преимущества этого символьного набора Вы увидите сразу, а другие несомненно оцените, поработав некоторое время. Мои друзья и знакомые давно с удовольствием пользуются именно этим набором, поскольку он достаточно универсален и текст замечательно выглядит на экране.

Для ввода этого символьного набора в память компьютера, надо набрать программу.

```
10 CLEAR 54599
20 LET N=64600: LET S=0
30 FOR X=N TO N+767
40 READ Y: POKE X,Y
50 LET S=S+Y:
60 NEXT X
70 IF S<>44655 THEN PRINT FLASH 1;"ERROR":STOP
80 POKE 23606,88: POKE 23607,251
90 FOR A=32 TO 127: PRINT CHR$ A;: NEXT A
100 SAVE "znak" CODE 64600,768
110 DATA
    000, 000, 000, 000, 000, 000, 000, 000,
    000, 024, 024, 024, 024, 000, 024, 000,
```

```
    000, 108, 108, 108, 000, 000, 000, 000,
    000, 108, 254, 108, 108, 254, 108, 000
120 DATA
    000, 024, 126, 088, 126, 026, 126, 024,
    000, 098, 100, 008, 016, 038, 070, 000,
    000, 048, 088, 048, 122, 204, 118, 000,
    000, 024, 048, 000, 000, 000, 000, 000
130 DATA
    000, 012, 024, 024, 024, 024, 012, 000,
    000, 048, 024, 024, 024, 024, 048, 000,
    000, 000, 108, 056, 254, 056, 108, 000,
    000, 000, 024, 024, 126, 024, 024, 000
140 DATA
    000, 000, 000, 000, 000, 024, 024, 048,
    000, 000, 000, 000, 124, 000, 000, 000,
    000, 000, 000, 000, 000, 024, 024, 000,
    000, 000, 006, 012, 024, 048, 096, 000
150 DATA
    000, 060, 102, 110, 118, 102, 060, 000.
    000, 024, 056, 024, 024, 024, 060, 000,
    000, 060, 102, 005, 060, 096, 126, 000,
    000, 060, 102, 012, 006, 102, 060, 000
160 DATA
    000, 012, 028, 044, 076, 126, 012, 000,
    000, 126, 096, 124, 006, 102, 060, 000,
    000, 060, 096, 124, 102, 102, 060, 000,
    000, 126, 102, 012, 024, 048, 048, 000
170 DATA
    000, 060, 102, 060, 102, 102, 060, 000,
    000, 060, 102, 102, 062, 006, 060, 000,
    000, 000, 024, 024, 000, 024, 024, 000,
    000, 000, 024, 024, 000, 024, 024, 048
180 DATA
    000, 000, 012, 024, 048, 024, 012, 000,
    000, 000, 000, 124, 000, 124, 000, 000,
    000, 000, 048, 024, 012, 024, 048, 000,
    000, 060, 102, 012, 024, 000, 024, 000
190 DATA
    000, 124, 206, 214, 222, 192, 124, 000,
    000, 060, 102, 102, 126, 102, 102, 000,
    000, 124, 102, 124, 102, 102, 124, 000,
    000, 060, 102, 096, 096, 102, 060, 000
200 DATA
    000, 124, 102, 102, 102, 102, 124, 000,
    000, 126, 096, 124, 096, 096, 126, 000,
    000, 126, 096, 124, 096, 096, 096, 000,
    000, 060, 102, 096, 110, 102, 060, 000
210 DATA
    000, 102, 102, 126, 102, 102, 102, 000,
    000, 060, 024, 024, 024, 024, 060, 000,
    000, 014, 006, 006, 102, 102, 060, 000,
    000, 102, 108, 120, 120, 108, 102, 000
220 DATA
    000, 096, 096, 096, 096, 096, 126, 000,
    000, 066, 102, 126, 102, 102, 102, 000,
    000, 102, 102, 116, 110, 102, 102, 000,
    000, 060, 102, 102, 102, 102, 060, 000
230 DATA
    000, 124, 102, 102, 124, 096, 096, 000,
    000, 060, 102, 102, 110, 110, 062, 000,
    000, 124, 102, 102, 124, 106, 102, 000,
    000, 060, 096, 060, 006, 102, 060, 000
240 DATA
    000, 126, 024, 024, 024, 024, 024, 000,
    000, 102, 102, 102, 102, 102, 060, 000,
```

```
    000, 102, 102, 102, 102, 060, 024, 000,
    000, 196, 198, 214, 214, 254, 066, 000
250 DATA
    000, 102, 060, 024, 024, 060, 102, 000,
    000, 102, 060, 024, 024, 024, 024, 000,
    000, 124, 012, 024, 048, 096, 124, 000,
    000, 030, 024, 024, 024, 024, 030, 000
260 DATA
    000, 000, 096, 048, 024, 012, 006, 000,
    000, 120, 024, 024, 024, 024, 120, 000,
    000, 024, 060, 090, 024, 024, 024, 000,
    000, 000, 000, 000, 000, 000, 000, 255
270 DATA
    000, 000, 220, 246, 245, 246, 220, 000,
    000, 000, 060, 102, 126, 102, 102, 000,
    000, 000, 124, 096, 124, 102, 124, 000,
    000, 000, 106, 106, 108, 108, 126, 006
280 DATA
    000, 000, 060, 108, 108, 108, 254, 196,
    000, 000, 126, 096, 124, 096, 126, 000,
    000, 000, 124, 214, 214, 124, 016, 000,
    000, 000, 062, 048, 048, 048, 048, 000
290 DATA
    000, 000, 102, 060, 024, 060, 102, 000,
    000, 000, 102, 102, 110, 118, 120, 000,
    000, 024, 102, 102, 110, 118, 102, 000,
    000, 000, 102, 108, 120, 108, 102, 000
300 DATA
    000, 000, 030, 054, 054, 054, 102, 000,
    000, 000, 066, 102, 126, 102, 102, 000,
    000, 000, 102, 102, 126, 102, 102, 000,
    000, 000, 060, 102, 102, 102, 060, 000
310 DATA
    000, 000, 126, 102, 102, 102, 102, 000,
    000, 000, 062, 102, 062, 054, 102, 000,
    000, 000, 124, 102, 102, 124, 096, 000,
    000, 000, 060, 102, 096, 102, 060, 000
320 DATA
    000, 000, 126, 024, 024, 024, 024, 000,
    000, 000, 102, 102, 062, 006, 060, 000,
    000, 000, 214, 214, 124, 214, 214, 000,
    000, 000, 124, 102, 124, 102, 124, 000
330 DATA
    000, 000, 096, 096, 124, 102, 124, 000,
    000, 000, 198, 198, 246, 222, 246, 000,
    000, 000, 060, 102, 012, 102, 060, 000,
    000, 000, 198, 214, 214, 214, 254, 000
340 DATA
    000, 000, 120, 012, 060, 012, 120, 000,
    000, 000, 198, 214, 214, 214, 255, 003,
    000, 000, 102, 102, 062, 006, 006, 000,
    060, 066, 153, 161, 161, 153, 066, 060
```

Числа в строках DATA напечатаны друг под другом для удобства чтения. Вы же можете набирать их так, как Вам удобно.

Теперь о "встраивании" загружаемого символьного набора в программу MF 09.

После того, как символьный набор будет записан на ленту, поступаем следующим образом. Разместим его непосредственно перед основным блоком кодов "MF09CODE" CODE 57328,8208 с адреса 57328-768-56560. Значение переменной CHARS тогда будет равно: 56560-256=56304.

Младший и старший байты CHARS:

56304-256*INT(56304/256)=240

INT (56304/256)=219

Далее делаем:

```
CLEAR 56559
LOAD "znak" CODE 56560,768
LOAD "MF09CODE" CODE 57328,8208
```

И записываем готовую программу на ленту:

```
SAVE "MF09 R/L" CODE 56560,8976
```

Теперь изменяем программу-загрузчик "MF LOADER", заменив CLEAR 57327 на CLEAR 56559.

Изменения, производимые в основной Бейсик-программе. Добавляем строки, переключающие основной и альтернативный символьные наборы

```
8 POKE VAL "23606",VAL "240": POKE VAL "23607", VAL "219": RETURN : REM RUS
9 POKE VAL "23606", NOT PI : POKE VAL "23607",VAL "60": RETURN : REM lat
```

Изменяем другие строки, подставляя в нужные места GO SUB 8 или GO SUB 9 и изменив начальный адрес и длину основного блока кодов:

```
1 GO SUB VAL "8": GO TO USR VAL "58285"
4020 GO SUB VAL "9": SAVE C$(TO VAL "10") DATA F$(): GO SUB VAL "8": GO TO USR R
4030 GO SUB VAL "9": SAVE C$(TO VAL "10") LINE VAL "4035": SAVE "MF09 R/L" CODE VAL
      "56560", VAL "8976": GO SUB VAL "8": GO TO USR R
4035 LOAD "MF09 R/L" CODE: GO TO PI/PI
```

Остальные строки оставляем без изменения.

Описанные выше способы русификации относились только к вводимым данным, однако более высокой степенью русификации является перевод на русский язык и замена текстовых сообщений в программе, которые печатаются по-английски.

3. Перевод программы на русский язык.

Пусть Вас не пугает то, что программа написана в машинных кодах. На самом деле все ненамного страшнее, чем в Бейсике. Причем для того, чтобы сделать полный перевод программы на русский язык, не обязательно даже умение пользоваться какими-либо специальными программами-мониторами типа MONS или другими. Не обязательно также знание шестнадцатеричной системы счисления. Необходимо только желание, немного терпения и аккуратности, да хоть немного знать английский язык или иметь словарь потолще. Неплохо также, если Вы поработали с программой какое то время, чтобы Вам понятен был смысл того или иного текстового сообщения.

Рассмотрим подробно процесс перевода программы на русский язык на примере программы MF 09. Приобретенный опыт Вы сможете использовать для перевода других программ.

В машинных кодах процедуры вывода текстовых сообщений на экран могут быть самыми различными, но в любом случае сама строка символов, выводимая на экран, находится внутри программы, надо только найти ее и изменить коды символов, находящихся там, занося другие значения хотя бы при помощи POKE.

Для работы можно воспользоваться любой программой-монитором, которая есть под рукой. А можно за несколько минут набрать нужную программу на Бейсике.

Ниже приводится описание такого специализированного монитора, который поможет нам находить текстовые сообщения в программе, а также несколько автоматизирует процесс замены текста на русский.

ПРОГРАММА-МОНИТОР

```
1 BORDER 7: PAPER 7: INK 0: CLS: GO SUB 6: GO TO 100
2 CLEAR 50000
3 LOAD ""CODE
4 GO TO 1
5 GO SUB 9: INPUT "FILENAME"; N$
6 SAVE N$ LINE 2: SAVE N$ CODE 56560,8976
7 STOP
8 POKE 23606,240: POKE 23607, 219: RETURN : REM rus
9 POKE 23606,0: POKE 23607,60: RETURN : REM lat
10 LET B=PEEK A: LET C$=CHR$ B
15 IF B<32 THEN LET C$="?"
```

```
20 RETURN
100 INPUT "ADDRESS: ";N
200 FOR A=N TO 65535
210 REM
220 GO SUB 10
230 INPUT (TAB 0;A; TAB 8; B;TAB 13;c$;TAB 23);C$: IF c$<>"" THEN POKE A,CODE C$
231 REM INPUT (TAB 0: A; TAB 8; B; TAB 13; C$; TAB 23); LINE C$: IF C$<>"" THEN POKE
      A,VAL C$
240 GO SUB 10
250 PRINT TAB 0; A; TAB 8;B; TAB 13; c$;
300 NEXT A
```

В строке 15 непосредственно перед знаком вопроса надо нажать "INV. VIDEO" (CAPS SHIFT+4), а сразу же после знака вопроса "TR. VIDEO" (CAPS SHIFT+3).

REM в строке 231 набран после набора всей строки, в последнюю очередь.

После того, как Вы наберете эту программу, сделайте RUN 2 и загрузите с магнитофона коды "MF09 R/L" с пристыкованным русско-латинским символьным набором. После окончания загрузки программа выполнит переключение на этот символьный набор и запросит адрес с которого хотим просматривать содержимое памяти компьютера. Пока введите "STOP" (SYMBOL SHIFT+A) и "ENTER" и поговорим подробнее о самой программе-мониторе.

Здесь вам уже знакомы некоторые фрагменты: строки 8 и 9 - переключатели шрифтов (в программе проставлены в нужных местах GO SUB 8 и GO SUB 9). Для сохранения результатов Вашего труда Вы периодически будете делать RUN 5 - это запись на ленту программы-монитора и кодов MF 09 в том состоянии, до которого Вы дошли. После того, как сделаете RUN 5, программа запросит имя файла для записи. Можете задавать 1, 2, 3... и т.д.

Строки 10, 15, 20 - это вспомогательная подпрограмма, присваивающая значения переменным в соответствии с содержимым ячейки памяти. Строка 15 предохраняет от вывода на экран управляющих символов, имеющих коды с 0 по 31. Если встретится такой символ, то на экране будет инверсно напечатан знак "?". Так Вы сможете различать управляющий символ и настоящий знак вопроса, которые тоже будут встречаться в тексте.

Строка 210 зарезервирована для организации поиска нужной информации, к ней мы еще обратимся позже.

Строки 230 и 231 очень похожи. В строке 231 стоит REM и она, таким образом выключена. Строка 230 выдает на экран адрес ячейки памяти, код, содержащийся в ней и символ, соответствующий этому коду; далее - в кавычках - ожидается ввод символа, код которого надо записать в эту ячейку. Это удобно при замене текстовых сообщений, так как ввод осуществляется непосредственно буквой, на место того символа, который мы видим, измененный текст тут же отображается на экране. Удобно также и то, что переключившись на загружаемый символьный набор программы "MF09 R/L", мы будем видеть текст в том виде, каким он будет в готовой программе. В режиме курсора [L] - печатаем русские буквы, а в режиме [C] (CAPS LOCK) - латинские. Действуют также регистры "EXT. MODE" и "GRAPH".

Вводить новый текст надо по одной букве, не забывая после каждого символа нажимать "ENTER", а также ставить "пробелы" между словами. Если Вы ничего не хотите менять, то просто нажимайте "ENTER" для перехода к следующей ячейке. Для того чтобы, остановить программу, надо нажать "КУРСОР ВЛЕВО" (SYMBOL SHIFT+5), затем "STOP" и "ENTER". Потом опять RUN или GO TO 100 для работы с новыми адресами. Если же Вы увидели, что допустили ошибку при вводе, то остановите программу, сделайте GO TO 200 - программа стартует без запроса с того адреса, который Вы вводили последний раз. Нажимая "ENTER", подойдите к тому месту, где допущена ошибка.

Теперь поэкспериментируйте с подготовленными программами, а в следующем выпуске ZX-РЕВЮ мы доведём до конца рассказ о том, как выполнить полную русификацию программы MASTERFILE 09.

СОВЕТЫ ЭКСПЕРТОВ

STALINGRAD

CCS 1989 г.

Эксперт Захаров М. Ю. г. Казань

Введение.

Стратегическая игра "STALINGRAD", разработанная фирмой CCS в 1989 году, представляет дальнейшее развитие игры "OVERLORD", расширяя ее возможности. Таким образом, фирма создала модели двух важнейших сражений второй мировой войны, переломивших ее ход на Западе и Востоке. Однако, если в "OVERLORD" Вы управляли войсками союзников, то в "STALINGRAD" Вам предлагается управление немецкими войсками. Можно не упоминать лишний раз о "холодной войне" и "образе врага" - одним словом, нам, конечно, это досадно и единственное, что можно сделать - подождать, пока отечественные программисты напишут программы, адаптированные под наше понимание истории.

Сражения на советско-германском фронте, уникальные своим огромным пространством и большим напряжением сил, драматичностью, наверняка еще послужат базой для сюжетов новых стратегических игр (например сражения за Москву, Берлин, сражение у озера Балатон, форсирование Днепра, освобождение Крыма, оборона Ленинграда, операция "Багратион", наступление в Маньчжурии), реализующих и советскую сторону в войне.

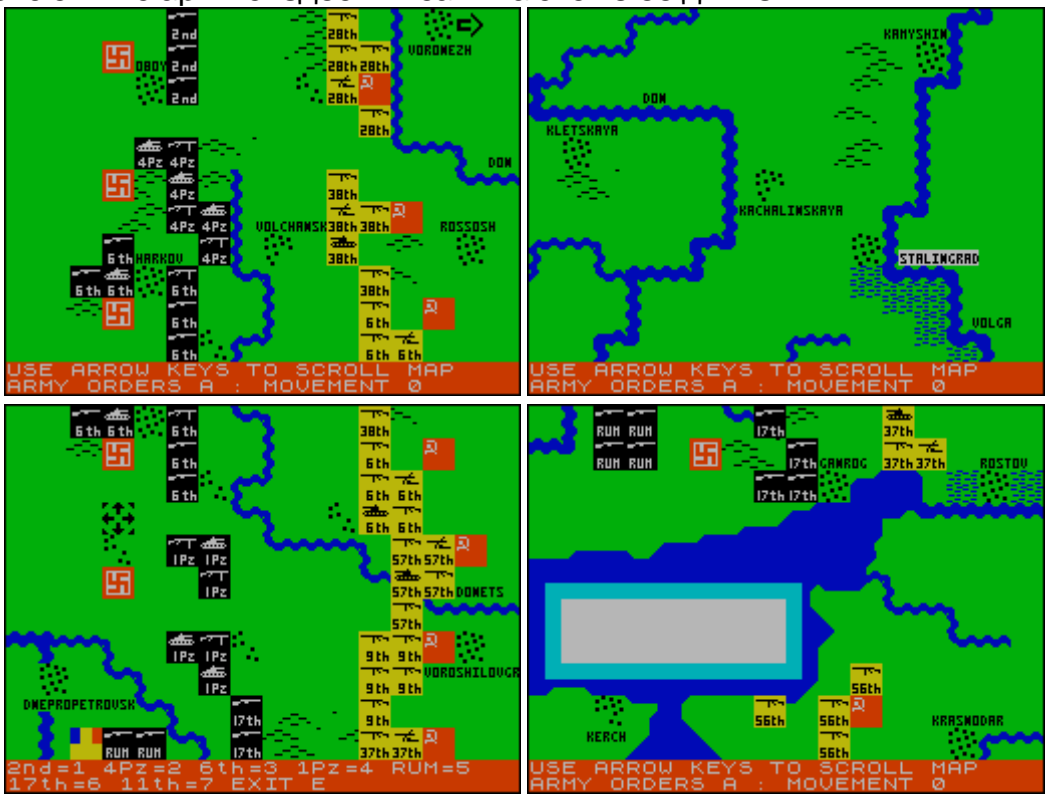
Вообще же, "STALINGRAD" - очень интересная игра, в ней реализованы такие возможности, как снабжение войск, корпусная организация армий, раздельное управление дивизиями (в OVERLORDe можно было управлять только армиями в целом). Боевые действия зависят от многих факторов (морального состояния войск, системы коммуникаций, рельефа местности и т. д.).

Она отражает события, происходившие на южном фланге советско-германского фронта с июня по декабрь 1942 года. В их результате Германия утратила, наконец, стратегическую инициативу, и перешла к обороне, как на Восточном, так и на Западном фронте.

К началу лета 1942 г. Германия была еще настолько сильна, что перешла в наступление на половине своего Восточного фронта. Цели этого наступления были весьма решительны: планировалось пересечь Кавказский хребет, завладеть нефтяными промыслами Азербайджана и нависнуть над английским Ближним и Средним Востоком, Ираном и Индией. Как сопутствующая цель рассматривался захват Сталинграда и перехват Волги, как важнейшей транспортной артерии, связывающей Советский Союз с предстоящим районом боевых действий, а заодно с бакинской нефтью и южными маршрутами поставок союзников. Красная Армия еще не обладала к этому моменту силами, достаточными для отражения такого наступления, хотя и пыталась его упредить. Попытки кончились для нас катастрофами под Харьковом и Керчью.

Наступая, немцы заняли Донбасс, затем перевалы Главного Кавказского хребта, Моздок, вышли к Сталинграду, но нараставшее сопротивление советских войск заставило Гитлера отложить свои далеко идущие планы, сосредоточившись на Сталинграде. Сталинград постепенно приковал все его ударные силы, и там они большей частью нашли себе могилу. Накопившая силы Советская Армия сама перешла в наступление, и 23 ноября вокруг Сталинградской группировки немцев замкнулось кольцо. Примерно этим периодом, судя по привлекаемым силам, завершается "STALINGRAD". Сталинградская битва же продолжалась. Советские войска отразили попытки немцев деблокировать окруженных, и к 2 февраля ликвидировали "котел".

Вы начинаете игру на позициях, примерно соответствующих линии фронта в июле 1942 г. В Вашем распоряжении 6 армий, одна из них в резерве - 3-я румынская (настроения в ней пониже, чем в немецких армиях). 11-я армия находится в Крыму. Вам противостоят слабо вооруженные и плохо оснащенные советские армии 1-го эшелона. Они пытаются перейти в наступление, но быстро терпят поражение. Этот этап вообще достаточно прост: Вы действуете вблизи баз снабжения, настроения в войсках высокие. Главная ударная сила в игре танковые дивизии. У Вас они сведены в 2 танковые армии, используйте их для быстрой победы. Попав хотя бы одной дивизией на Таманский полуостров, Вы активизируете свою 11-ю армию: здесь высаживаются 3 ее дивизии.



Однако кончается лето, начинается распутица, линии Ваших коммуникаций растягиваются. Вам приходится прикладывать все большие усилия для овладения новыми пунктами для баз снабжения. Численность ваших войск сократилась, а выделяемые верховным командованием пополнения скудны. Настроение в войсках падает. А тем временем из глубины Советского Союза подходят и разворачиваются свежие и хорошо оснащенные армии, имеющие большое количество танков и мотопехоты. (Обратите внимание на стрелки на краях карты военных действий: они обозначают полосы, в которых происходит это выдвижение), теперь все далеко не так просто!

Управление игрой

После загрузки программа запрашивает уровень сложности, предлагает загрузить отложенную партию. После этого Вы видите карту с расположением войск. Карту можно скроллировать клавишами курсора. В нижней части экрана находятся сменяющие друг друга меню.

Каждый ход состоит из 2-х этапов: отдания приказов и собственно перемещения войск. В начале каждого хода программа предлагает записать ситуацию (SAVE GAME - S),

отдать приказы (ARMY ORDERS - A) и перейти к их исполнению и перемещению войск (MOVEMENT - O).

Ваши армии состоят из 3-х родов войск: пехоты, мотопехоты, танков (каждый последующий род сильнее и маневреннее предыдущего). Аналогична структура советских войск, хотя, символы, обозначающие род войск различны.

Обратите внимание на символы с 4 стрелками. Это базы снабжения. Они могут располагаться лишь в населенных пунктах, являющихся узлами дорог. По мере продвижения Ваши поиски будут опираться на сеть этих баз. База появляется в очередном занятом Вами пункте, когда он оказывается у Вас в тылу.

Нажав в главном меню "A", Вы попадаете в меню принятия решений. Здесь Вы можете отдать приказы (ORDERS - O), просмотреть состояние сил обоих сторон (DETAIL - D) и их расположение (TERRAIN - T). Каждой армии соответствует цифровая клавиша, с помощью которой Вы к этой армии обращаетесь. Перейдя в режим ORDERS, Вы отдаете приказ выбранной армии, указывая ей рубеж, который она должна занять, и характер действий (наступление ATTACK - A, оборона DEFEND - D). Большинство Ваших армий, кроме 2-й и 11-й, включают по 6 дивизий, сведенных в 2 корпуса (3 дивизии в каждом), 2-я и 11-я армии состоят из одного корпуса каждая. Вы указываете последовательно положение центра, правого и левого флангов сначала одного, затем другого корпуса армии. Дивизии при этом помечаются символами "X", "R", "L" - то есть центр, правый и левый фланги корпуса. Таким образом, Вы можете указать желаемое место для каждой дивизии, что, по сравнению с OVERLORDом, большое удобство.

Прелесть же "самовольных" последующих действия дивизии, обусловленных принадлежностью ее к корпусу и армии, STALINGRAD унаследовал у OVERLORDa. Заданный рубеж может быть достаточно далеким: соединение будет выполнять свою задачу решая тактические вопросы с определенной долей самостоятельности. Однако если оно подверглось удару противника и стало откатываться назад, приказ (если Вы не отменяете его) лучше продублировать.

В режиме "DETAIL" Вы можете просмотреть численность войск обоих сторон. Вы увидите и настроение своих частей:

EXLT - excellent - превосходно
V. G. - very good - очень хорошее
GOOD - хорошее
FAIR - благоприятное
POOR - ухудшенное
LOW - низкое
ABYS - отчаяние

От настроения зависит эффективность ваших войск, их потери. В режиме "TERRAIN" программа выделит расположение выбранной вами армии. Ее дивизии закрашиваются одноцветными квадратиками без всяких надписей, что доставляет большое неудобство, поэтому этим режимом лучше не пользоваться. Выбравшись из младших меню в главное посредством многократных нажатии "E" (Exit), Вы и можете привести войска в движение, нажав "O" (MOVEMENT). Программа поочередно показывает движение и нанесение ударов обеими сторонами (удары наносят, разумеется, только наступавшие части). Вы видите потери, которые несут сражающиеся дивизии в процентах. Между действиями немецких и советских войск Вам предлагается пополнить свои части. В рамке-меню, где-то в районе Азовского моря, перемещая указатель клавишами "P" и "Q", Вы выбираете очередную пополняемую армию, нажимая "A". Если армия отрезана от баз снабжения, доступа к ней нет. Затем - род войск пополнения (фиксация - "T") и его численность (SET VALUE - V). В этом меню Вы видите размеры имеющихся у Вас резервов:

ARM - танки
MECH - мотопехота
INF - пехота (инфантерия)

На протяжении нескольких первых ходов пополнения Вам не выделяются.

В ходе боевых действии некоторые дивизии не просто несут потери, но исчезают совсем: они могут быть расформированы (Unit disband) или разгромлены (Unit routs). Остатки расформированной дивизии пополняют другие дивизии армии. Вместо исчезнувших дивизий можно сформировать новые из пополнения любого рода войск. Таким образом, в состав пехотных армий можно включить танковые и мотопехотные дивизии, и наоборот (но не стоит распылять силы).

Формирование новой дивизии произойдет, если передать армии большое пополнение - 100 или 200.

Армия не может превзойти своих первоначальных размеров: она не может включать больше 6 дивизий, а дивизия не может иметь более, чем 100% состав.

Когда все Ваши резервы будут исчерпаны, программа выдаст Вам свое заключение о том, кто же победил, об устойчивости победы и сообщит количество выживших в результате Ваших действий у обеих сторон. Заключение вполне может не оправдаться, Вы можете продолжить игру, ответив "Y" на соответствующий вопрос. Правда, игра с каждым ходом будет все больше напоминать эндшпиль шахматной партии, когда на доске остается по 2-3 фигуры.

Некоторые детали

Пусть Вас не удивляет, что последние советские армии первого эшелона стремительно расформировываются, выйдя из соприкосновения с Вами. Они очищают место для вступления на сцену II эшелона, вливаясь в его дивизии.

Обратите внимание на то, что программа перемещает армии последовательно, по их расположению с севера на юг вдоль линии фронта. Поскольку порядки одной армии плохо проницаемы для другой, это правило стоит учитывать, если Вы не хотите, чтобы Ваши войска перепутались.

Компьютер скрывает расположение своих частей, не находящихся в контакте с Вашими. Но, если Вы вызовете данные о них в режиме "DETAIL" или "TERRAIN", он проскроллирует карту и укажет приблизительный район их расположения, хотя, конечно, цветом их не выделит.

Следите за целостностью своих коммуникаций. Снабжение армии может быть прервано по нескольким причинам: она далеко оторвалась от баз снабжения, советские войска совершили рейд по Вашим тылам (нередко компьютер направляет Вам в тыл танковые бригады), наконец, Вы сдали свою базу, отходя. Лишившись снабжения, армия сразу теряет в эффективности: падают маневренность, настроение в войсках, растут потери и их нечем восполнять. Для разгрома такой армии достаточно нескольких ходов. Чтобы восстановить снабжение, надо заново создать всю систему коммуникаций, для чего армия должна вплотную подойти к любой базе снабжения. Нередко приходятся довольно долго отступать.

Уровни сложности игры

В начале игры программа предлагает выбрать уровень сложности (ENTER GAME LEVEL). Их три: (1-3).

Принципиальных различий между ними нет, но из-за многочисленных мелких сложностей игра на 3 уровне значительно труднее.

Выгрузка файлов на ленту

Игра достаточно продолжительна, полный ее цикл длится около 8 часов, поэтому Вам наверняка придется записывать партию на ленту, и не один раз. Программа предлагает сделать это в начале каждого хода. Перед выдачей информации на магнитофон выдается предупреждение (Press any key). Записываемый файл имеет длину более 16 килобайт, в нем сохраняются не только положение войск, но и отданные Вами приказы.

Должен предупредить об одной детали. Бывает, что программа выводит сообщение "Unit disband" уже после записи партии. В этом случае на моем компьютере версии "Балтика" записанный файл оказывается сбойным. После его загрузки игра "виснет".

Приходится дожидаться следующего хода и записывать партию заново.

Программа предлагает загрузить отложенную партию только один раз, в начале игры, после ввода уровня сложности.

STALINGRAD принадлежит к тем немногим играм, которые не могут быть запущены заново без перезагрузки из-за большого объема информации, характеризующей ситуацию. Поэтому нельзя прервать неудачную партию иначе, как только перезагрузив программу. По достижении победы одной из сторон программа тоже предлагает считать себя заново с ленты.

Замечания

События, реализуемые игрой, начальное положение сторон, состав сил, стартовые действия, завязывающие ситуацию, конечно, лишь напоминают реальные события 50-летней давности. Все мои попытки повторить Сталинградскую битву не привели к успеху. Советские войска в игре не включают ни 5-й танковой армии, ни танковых и механизированных корпусов, танки и мотопехота рассеяны по общевойсковой армиям, 2-я немецкая армия в действительности была венгерской. 11-я армия вполне самостоятельно переправилась из Крыма летом 1942 г. Существует и много других немаловажных деталей, отличающих Сталинград от STALINGRADa.

Но, как самостоятельная игра, STALINGRAD очень интересна и наверняка привлечет ваше внимание.

FLIGHT SIMULATION[1]

Psion 1983 г.

Перевод фирменной инструкции "ИНФОРКОМ", 1987

В последние годы пилотов все чаще обучают, используя имитаторы. Даже на маленьком компьютере можно реально, в реальном времени, отобразить многие особенности полета: динамику самолета, навигацию, инструментальное обеспечение и т. д. "Имитатор полета" включает в себя эти эффекты и представляет модель полета маленького двухмоторного винтового самолета.

Аспекты полета

Органы управления самолетом включают в себя: штурвал, закрылки, хвостовой руль и регулятор мощности двигателей. Движение штурвала вперед и назад вызывает отклонение горизонтальных рулей и, соответственно, перемещение самолета вниз или вверх.

Аэродинамика самолета чрезвычайно сложна. Управление каким-либо органом вызывает, как правило, не одно последствие. Например, элероны не только вызывают крен самолета, но и создают дополнительные боковой воздушный поток, вызывавший поворот самолета. Всему этому можно научиться, работая с имитатором.

Положение самолета и характер полета отображаются на приборной панели в кабине

[1] Последующие версии носят название Flight Simulator (Прим. NUK)

пилота. Пилот должен руководствоваться показаниями этих приборов при выведении самолета на линию, с которой он совершит посадку с требуемой скоростью, под необходимым углом. Обычно правильный угол посадки составляет 3 град., т.е. высота должна быть 6000 футов на расстоянии 20 миль, 3000 футов на расстоянии 10 миль и 1000 футов на расстоянии 3 мили от ВПП. Руль направления также может несколько изменять курс самолета. При движении по ВПП именно им регулируется направление движения самолета.

В имитаторе Вы можете приземляться на одном из двух аэродромов, взлетать с них, ориентироваться с помощью радиомаяков или по местности. Через кабину Вы видите светлое небо и темную землю, огни взлетно-посадочной полосы в трехмерной перспективе и объекты на земле: озеро и т.п. Экран можно переключить также на изображение навигационной карты, на которой показаны детали местности: ВПП, радиомаяки и пр. После загрузки программы в машину на экране появляется меню с вопросом: "Что Вы хотите отрабатывать?" - взлет, гладкий полет или посадку. Нажмите, соответственно, 1, 2 или 3. После этого Вас спросят, нужен ли вам учет ветра. Отвечайте "Y" только если Вы опытный пилот и можете справиться с внезапными порывами ветра, в противном случае - "N".

Приборная панель

На нижней части экрана изображена приборная панель кабины пилота. Пять циферблатов слева-направо это:

1. Система инструментальной посадки (ILS).
2. Датчик скорости.
3. Оборудование наведения на радиомаяки (RDF).
4. Высотомер.
5. Индикатор скорости высоты (ROC).

RDF - это большой циферблат в центре панели. Здесь изображен символ самолета, он указывает направление полета самолета. Цифровой указатель дает направление полета в градусах компаса. RDF - это самая необходимая навигационная система, в любое время полета самолет связан с одним из радиомаяков. Положение маяка, к которому в данный момент подключился самолет, изображается светлым крестиком на окружности циферблата RDF. Если Вы хотите двигаться на данный маяк, Вам надо развернуть самолет так, чтобы положение маяка совпадало с "12 часами" на циферблате RDF.

Датчик скорости находится справа от RDF. Малая стрелка указывает высоту в тысячах футов, а большая в сотнях футов.

Индикатор ROC - это индикатор скорости подъема. Он измеряет вертикальную скорость самолета в единицах 1000 футов/мин. Когда стрелка выше нуля, самолет идет вверх и наоборот.

Индикатор POWER - "мощность" - измеряет тягу моторов, тяга двигателей возрастает при увеличении POWER, но на большой высоте падает.

FUEL - "топливо" - наличие топлива в баке.

FLAPS - "закрылки" - показывает угол выдвижения закрылков.

GEAR - "шасси" - имеет два состояния. Шасси убрано - красный цвет; шасси выпущено - зеленый цвет.

BCN, RGE, BRG - информация по маяку, к которому привязан в данный момент самолет. BCN - позывной маяка. RGE - расстояние до маяка в милях. BRG - курс в градусах относительно маяка. ILS -система инструментальной посадки. Эта система для наведения самолета на аэродром. Радиобуй, размещенный в начале ВПП, излучает сигнал, положение которого видно на экране ILS, как светящееся пятно. Когда самолет приближается к аэродрому с правильного курса, светящееся пятно будет в центре экрана.

Ra - радиовысотомер, это часть системы ILS. Радиосигнал, отраженный от земли, позволяет замерить высоту от земли до колес самолета. Замер в футах. Это точный отсчет; применяется при посадке.

Управление самолетом

Управление влево-вправо-вверх-вниз с помощью курсора - клавиши 5, 6, 7, 8.

Управление горизонтальным рулем - Z - поворот влево. X -вправо. Им же управляется самолет при движении со ВПП.

Тяга двигателя: P – повышение, O - понижение.

Закрылки:

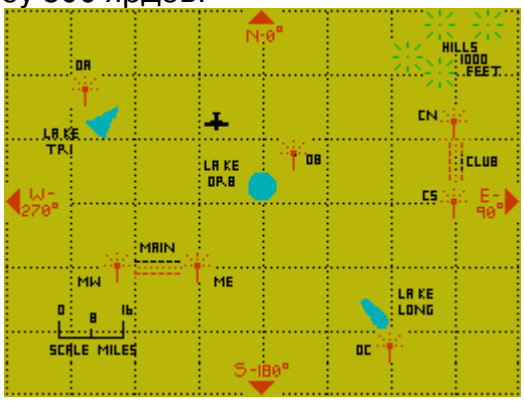
F - увеличение выхода закрылков; D - втягивание закрылков.

С убранными закрылками скорость отрыва самолета - 80 узлов, с выпущенными закрылками - 60 узлов. Увеличение выхода закрылков при большой скорости полета может привести к разрушению крыльев.

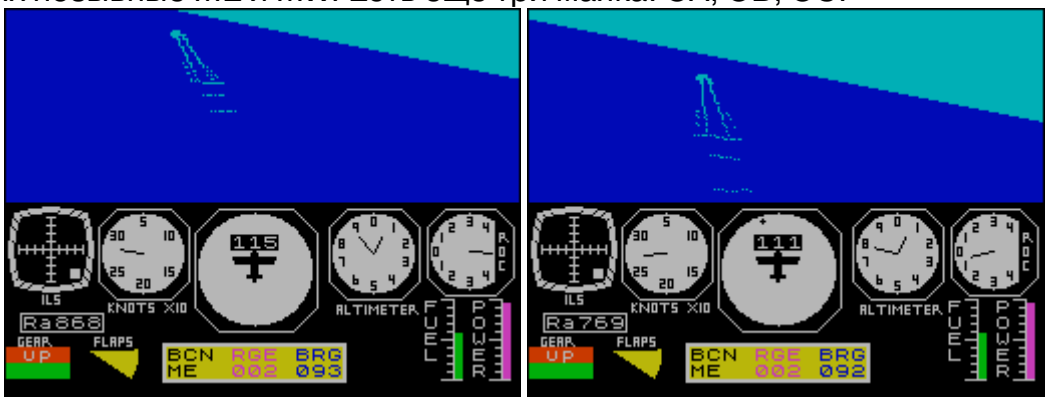
Шасси: выпуск и втягивание - клавишей G. Нельзя выпускать шасси при большой скорости, т.к. оно может заклинить или оторваться.

"Радиомаяки": - чтобы подключиться к другому маяку, нажмите клавишу B. Пока Вы будете держать эту клавишу, изменяется маяк, к которому привязан самолет.

Карта: - нажав кнопку M, Вы получите на экране изображение навигационной карты. На карте изображены два аэродрома: один международный аэропорт, MAIN и один маленький клубный аэродром CLUB. MAIN имеет длинную ВПП длиной более мили и поэтому посадка на него для небольшого самолета несложна, CLUB - это маленький местный аэродром и имеет полосу 800 ярдов.



Карта изображает также положение маяков и различных наземных объектов. Около главного аэропорта есть два маяка, расположенные на расстоянии трех миль от начала и от конца полосы. Их позывные ME и MW. Есть еще три маяка: OA, OB, OC.

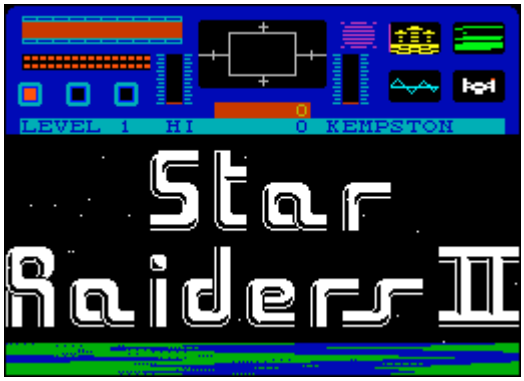


Навигация

Самая трудная часть полета это подход к аэродрому и посадка. На достаточно большой высоте Вы можете экспериментировать с органами управления, меняя скорость и направление полета, не заботясь о навигации. Если же Вы хотите совершить посадку, Вам надо положить самолет на правильный курс и подойти к аэродрому с правильным углом посадки. Это трудная задача и она требует больших усилий и тренировки, пока Вы совершите посадку правильно.

STAR RAIDERS

"Звездные рейнджеры"
ATARI 1987



Эксперт Порядин С. В.
Марийская ССР

Игра STAR RAIDERS 2, разработанная фирмой ATARI, органично объединяет в себе несколько игровых жанров. В этой игре Вы можете проявить себя, как стратег в планировании операции против захватчиков из другой звездной системы, так и как пилот боевого космического корабля в схватках с эскадрами кораблей захватчиков.

Боевые действия ведутся как в открытом космосе в двух звездных системах: Вашей "CELOS" и вражеской "PROCYON", так и над поверхностями планет, на которые высадились вражеские корабли. В Вашей звездной системе вокруг центрального светила вращаются три планеты, одна из которых имеет спутник. Все планеты и спутник заселены и основное население сконцентрировано в городах. Уничтожение городов является главной целью захватчиков.

В Вашей звездной системе также находятся несколько космических станций, на которых отремонтируют Ваш корабль, дозаправят его горючим и боеприпасами.

Враги

Они прилетают из своей звездной системы. Их корабли никогда не появляются в одиночку, они все время летают эскадрами, в состав эскадры входит несколько флайеров (иногда их число больше дюжины), два или три крейсера и один флагманский корабль.

Флайер имеет легкое вооружение и может быть сбит единственным попаданием, но флайеры нападают сразу вдвоем или втроем, так что неопытному пилоту будет поначалу очень "жарко". При уничтожении всех флайеров в эскадре противника наступает черед мощных и опасных крейсеров. Они имеют очень мощное оружие и могут быть поражены одним или несколькими попаданиями бомб. Смена оружия в вашем корабле происходит автоматически, как только на Вас нападет крейсер или флагманский корабль. Но космические битвы - это не то, для чего предназначены крейсера (хотя они очень маневренны). Основная их цель уничтожение городов на Ваших планетах.

Флагманский корабль гораздо мощнее вооружен чем крейсер, но опасности для Ваших планет он не несет. Правда, за уничтожение флагмана Вам будет начислена большая сумма очков. Но основная Ваша цель - это защитить города на планетах.

Вражеские эскадры также могут атаковать станции. Если врагов вовремя не отогнать

от них, то станции могут быть уничтожены и Вы останетесь без баз. На нижнем уровне игры, когда у Вас имеется три станции, потеря одной из них не несет больших неудобств. Но на более высоком уровне это может обернуться для Вас катастрофой.

Уничтожая вражеские эскадры, невозможно добиться победы, так как из вражеской звездной системы в Вашу будут лететь все новые и новые экспедиции. Чтобы одержать полную победу над врагом, нужно уничтожить его базы. Базы врага находятся на трех планетах его звездной системы. Уничтожить базу противника можно метким попаданием атомной бомбы. Для включения системы бомбометания при полете над поверхностью планеты необходимо нажать клавишу "W". Тогда внизу обзорного экрана появится прицел для бомбометания. При приближении к базе на медленной скорости нужно вывести ее изображение под прицел и нажать "огонь". За один заход невозможно уничтожить все базы противника, т.к. у Вас ограничен запас бомб, так что придется возвращаться на станцию для пополнения боезапаса.

Запуск игры

После загрузки программа предложит вам выбрать тип джойстика и игровой уровень от "1" до "3". После нажатия "S. SHIFT" Вы приступаете непосредственно к игре.

После запуска игры Ваш корабль находится на одной из планет своей звездной системы. Он сразу же будет атакован вражескими флайерами, но это лишь разведчики врага, и никакой опасности для городов они не представляют. Поэтому Вы можете сразу отлететь от планеты, нажав "SPACE", выбрав курс и затем нажав клавишу "огонь". Теперь, в более спокойной обстановке, можете внимательно рассмотреть находящееся на экране Вашего дисплея изображение.

Экран

Весь экран можно условно разделить на две части. В верхней, меньшей его части, расположена приборная доска вашего корабля. На ней находятся несколько индикаторов, показывающих состояние его систем.

В левой верхней части расположен индикатор энергии, чуть ниже этого индикатора расположено два ряда прямоугольных меток - это счетчик атомных бомб, имеющихся в Вашем распоряжении. Ниже этого счетчика находятся три индикатора, показывающих, какое оружие в данное время Вами задействовано.

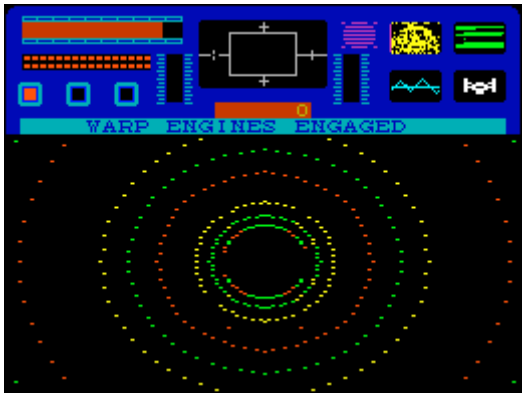
В центре приборной доски находится экран радара, который позволит ориентироваться в сложной обстановке боя. Если нажать клавишу "T", то изображение на нем сменится изображением Вашего корабля, на котором условными метками будут указаны системы, пострадавшие от вражеского огня. Если контур корабля на экране радара обнесен точками, это значит, что включено защитное поле, в местах, где поле пробито огнем вражеских кораблей точки отсутствуют или мигают.

По обе стороны экрана радара находятся две вертикальные шкалы показывающие температуру лазеров, ниже расположен счетчик очков, заработанных Вами.

Остальная информация на приборной доске не несет большой смысловой нагрузки.

Ниже приборной доски расположен экран Вашего корабля, на котором и показывается обстановка по курсу. Между экраном и приборной панелью находится информационная строка, на ней в ходе игры будут появляться сообщения о различных действиях врага и о критическом уровне энергии в Вашем корабле.

Если во время игры нажать клавишу "SPACE", то на экран корабля будет спроецирована карта звездной системы, в которой он находится. При этом внизу экрана расположена информация о планете, на которую Вы направили указатель гиперперехода или на который Вы находитесь.



Управление

Управление кораблем не очень сложное и осуществляется с помощью выбранного джойстика. При полете над поверхностью планеты движением ручки джойстика вверх-вниз регулируется скорость Вашего корабля.

Клавиша "S" служит для вкл/выкл защитного поля.

Клавиша "W" служит для переключения стрельбы лазером на бомбометание и обратно. На то, какое оружие используется, указывает соответствующий индикатор.

Клавиша "T" служит для переключения экрана радара на индицирование состояния систем корабля. При повторном нажатии включает радар.

Клавиша "P" служит для временного останова игры.

Клавиша "C. SHIFT" заканчивает игру и выводит на экран заставку.

Для гиперперехода необходимо вывести на экран карту звездной системы, в которой Вы находитесь, и, манипулируя джойстиком, нужно установить указатель гиперперехода на любой объект в этой системе. Если после этого нажать "огонь", не выходя из "карты", то переход будет сделан туда, куда Вы указали.

Перелет в другую звездную систему можно осуществить, предварительно установив указатель гиперперехода на ее символьное изображение в углу карты и нажав клавишу "огонь".

Энергия

Энергия расходуется на поддержание защитного поля во время обстрела Вашего корабля вражеским и во время гиперпереходов. При полном расходе запаса энергии корабль погибает.

Теперь несколько советов, которые могут быть Вам полезны.

Старайтесь не допустить высадки вражеской эскадры на планету в Вашей системе, так как сразу после этого крейсера начнут уничтожать города, да и сбить крейсер над планетой значительно сложнее.

Не старайтесь полностью уничтожать эскадру в космосе. Достаточно уничтожить все крейсера. Флагманский корабль, оказавшись в одиночестве, не представляет серьезной угрозы, а схватка с таким мощным кораблем после боя с крейсерами, когда в Вашем корабле мало энергии, может оказаться роковой.

При нападении на станцию вражеской эскадры, ее изображение на карте начинает мигать и Вам нужно спешить к ней на выручку, но если у Вас есть более срочные дела, например уничтожение высадившихся на планету врагов, то разборку с врагами, напавшими на станцию, можно отложить. Некоторое время станция может продержаться сама. Однако, следует помнить, что на более высоких уровнях игры это время значительно сокращается, да и станций там меньше (на третьем уровне - всего одна станция). Так что рисковать не советуем.

Из этой войны Вы выйдете победителем, если уничтожите все базы противника и все

эскадры вражеских кораблей, но при этом у Вас должен остаться хотя бы один город на любой из планет. Если же враги уничтожат все Ваши города, то это будет поражение. Естественно, Вам будет засчитано поражение, если враги смогут сбить Ваш корабль. На первых порах это и будет происходить чаще всего, но, потренировавшись и накопив боевой опыт, Вы станете серьезным препятствием на пути космических агрессоров.

THE TRAIN

("Поезд")
Accolade 1988.
Эксперт Порядин С. В. Марийская ССР

The Train представляет собой очень оригинальную программу-иммитатор. В этой игре Вам будет предложено испытать свои силы в качестве машиниста паровоза. Кроме того, в ней присутствуют элементы других игровых жанров, в том числе и стратегических игр.

Краткий сюжет

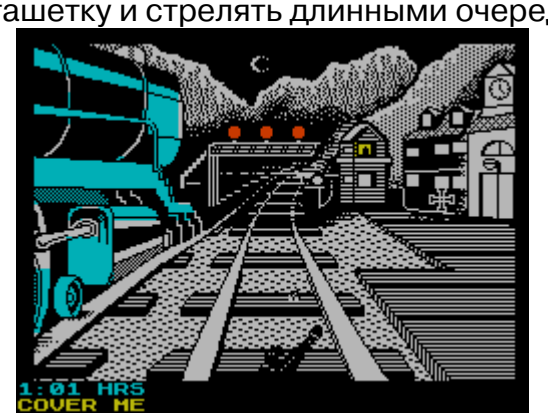
Представьте себе, что Вы находитесь в центральной части Франции в 1944 году. Войска союзников высадились на западном побережье и ведут бои с войсками вермахта. Вы командир одного из отрядов сопротивления, и Вашему отряду дано задание отбить на станции METZ (г. Мец) бронепоезд и провести его к войскам союзников.

От Вашего умения ориентироваться в сложной обстановке зависит успех данного предприятия. Вам нужно спланировать маршрут движения бронепоезда по железным дорогам Франции. По пути придется прорываться через станции и железнодорожные мосты, захваченные фашистами. Их нужно отбить у врага. В пути на бронепоезд будут совершать налеты самолеты Люфтваффе и Вам придется от них отбиваться. Ну и кроме всего этого, от Вас потребуется непростое умение справиться с такой сложной машиной как паровоз. Ведь кроме опасностей, подстерегающих Вас на пути, могут произойти большие неприятности связанные с неумением справиться с органами управления в кабине паровоза. Вы можете погибнуть от взрыва котла или, наоборот, бронепоезд остановится из-за нехватки давления пара, если Вы неправильно рассчитаете маршрут движения и у Вас кончится уголь или вода. У паровоза также могут выйти из строя тормоза, и игра на этом закончится.

Настройка программы

После запуска программа предложит выбрать джойстик и сразу после этого Вы приступите непосредственно к игре.

В начале игры Вы находитесь с пулеметом на путях возле паровоза. Ваша задача - прикрыть своего помощника Ле Дюка, который направился к стрелке, чтобы перевести ее. Вам нужно уничтожать немецких солдат, силуэты которых будут появляться в окнах зданий расположенных поблизости. Запас патронов у Вас неограниченный и можно порекомендовать не отпускать гашетку и стрелять длинными очередями.

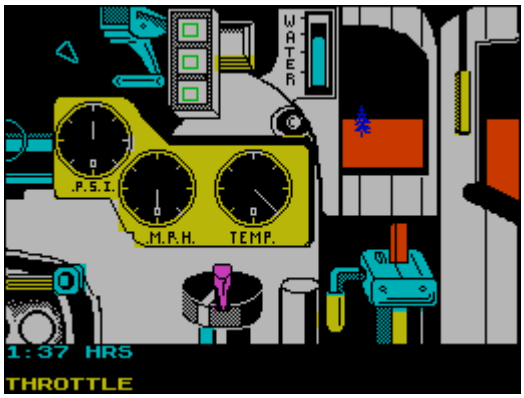


Как только Ле Дюк доберется до семафора, Вам будет предложено выбрать путь, по которому пойдет бронепоезд. Путь выбирается с помощью клавиш управления или джойстика. Затем нужно прикрыть возвращение Вашего помощника. Как только он

доберется до паровоза, Вы автоматически оказываетесь на рабочем месте машиниста и приступаете к основной части игры.

Экран

Перед Вами расположено несколько приборов и органов управления паровозом. На приборах показывается: скорость паровоза (MPH), давление пара (PSI), его температура (TEMP) и уровень воды (WATER).



К органам управления относятся: тормоза (BREAKE), рычаг реверса (FORWARD/REVERSE LEVER), заслонка сброса пара (STEAM BLOWOFF), клапан регулировки давления (THROTTLE), кроме этого перед Вами в кабине находятся крышка топки и ручка свистка.

Управление

Управление производится с помощью джойстика и некоторых клавиш на клавиатуре.

В кабине паровоза Вы можете перемещать с помощью джойстика стрелку, указывающую на какой-либо орган управления, с помощью джойстика можно и манипулировать им.

Чтобы поднять температуру и давление пара, необходимо открыть крышку топки и подбросить туда уголь (рукоятка джойстика вправо). Скорость движения бронепоезда регулируется рычагом управления давлением пара и тормозами. Чем больше Вы потянете на себя рычаг регулировки давления, тем больше пара будет поступать в цилиндры и тем быстрее будет двигаться бронепоезд. Для остановки нужно перекрыть поступление пара в цилиндры, толкнув рычаг до упора от себя и затормозить, повернув рукоятку тормоза. Приводить тормоз в действие лучше кратковременными импульсами по 2-3 секунды, иначе он быстро выйдет из строя, при закрытой заслонке начинает расти давление пара в котлах и, чтобы удержать его в норме, необходимо стравливать пар с помощью заслонки (STEAM BLOWOFF).

Из кабины паровоза Вы можете перейти к одному из зенитных пулеметов, расположенных спереди и сзади бронепоезда. Это необходимо при отражении атак немецких самолетов. Переключение осуществляется нажатием клавиш "1" или "2": возврат в кабину - "3". О налете вражеской авиации Вас проинформирует компьютер, он же выдаст сообщение, если давление пара будет превышать норму или будет слишком низким. Компьютер проинформирует Вас и в том случае, когда запасы угля или воды будут подходить к концу.

Включив "Таблицу состояния" с помощью клавиши "5", можно узнать о состоянии различных систем бронепоезда, из таблицы можно узнать о процентном износе тормозов, котла, брони, и также о количестве оставшегося угля. Выход из таблицы - нажатием клавиши "5".

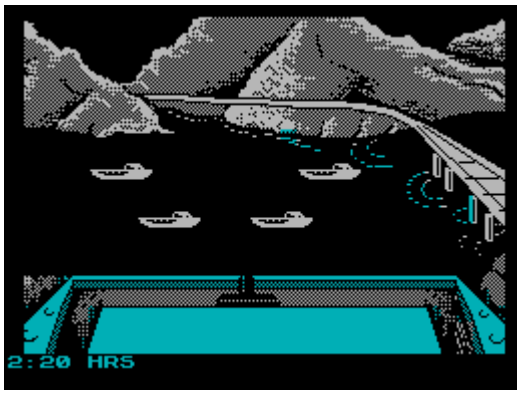
С помощью клавиши "4" можно вывести на экран карту сети железнодорожных путей запада Франции, на которой будут отмечены все станции, мосты, развилки и также указано положение Вашего бронепоезда. По этой карте Вы узнаете о приближении бронепоезда к станции или мосту, а также спланируете дальнейший путь состава по оптимальному маршруту. Другие клавиши:

"SPACE" - пауза.

"R" - выход из игры.
"S" - вкл/вкл звука.

Мост (BRIDGE)

На мосту, захваченном союзниками или отрядами сопротивления (о чем свидетельствует изменение его цвета на карте), Вы можете не останавливаться. Но как только попытаетесь проехать через мост, который контролируется немцами, без остановки, то будете немедленно уничтожены.



Поэтому при подходе к мосту необходимо замедлить скорость до минимума и, когда расстояние до моста сократиться до нуля, о чем Вас проинформирует компьютер, полностью остановить бронепоезд. В этот момент Вы автоматически окажетесь в орудийной башне и должны будете расстрелять немецкие речные суда, которые тоже будут отвечать огнем. Как только все суда будут потоплены, путь станет свободен и можно двигаться дальше.

Станция (STATION)

На станциях, захваченных врагом, можно не останавливаться, но после особенно длинных перегонов или перед ними необходимо заправить баки водой и загрузить уголь, это можно сделать лишь отбив у врага станцию. Для захвата станции нужно действовать аналогично случаю с мостом. После остановки Вы окажетесь на месте пулеметчика и Ваша задача - подавить сопротивление немцев. После захвата станции Вам предложат ознакомиться со сводкой германского командования. Нажав клавишу "Огонь", Вы увидите следующее меню:

- захват следующей станции;
- захватить следующий мост;
- сделать ремонт;
- ничего не делать.

Вы можете отправить соответствующее послание союзникам и получите ответ, в котором будет указано, в какое время произойдет данное событие (текущее игровое время указывается на циферблате в углу экрана).

После этого Вы можете продолжать движение.

Развилка или перекресток (SWITCH)

Повернуть на другие пути можно только при полной остановке бронепоезда на развилке при помощи рычага реверса.

Внимание!

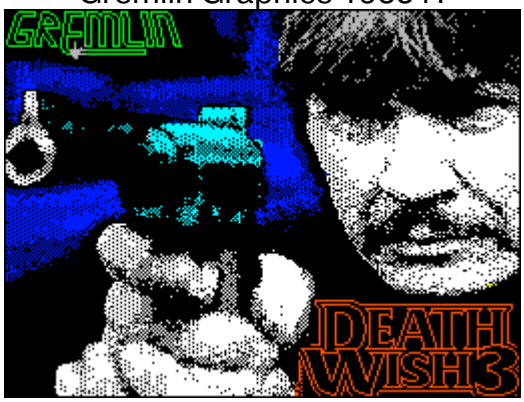
Не трогайте рычаг до полной остановки бронепоезда.

На первых порах будет очень трудно справиться с управлением бронепоездом, но потренировавшись некоторое время, можно приобрести опыт и добиться больших успехов в этой необычной компьютерной игре.

DEATH WISH 3

("Жажда Смерти - 3")

Gremlin Graphics 1988 г.



Эксперт Троекуров В. И. г. Киев.

Фирма GREMLIN GRAPHICS выпустила свою программу на рынок вслед за появлением третей серии всемирно известного кинофильма "Жажда Смерти", главную роль в которой играет очень симпатичный и любимый многими мужественный Чарли Бронсон.

Те, кто не знаком с этим киносериалом, нередко полагают, что где-то существуют программы DEATH WISH 1 и DEATH WISH 2. Но к сожалению это не так. Цифру "три" программа получила от третьей серии фильма.

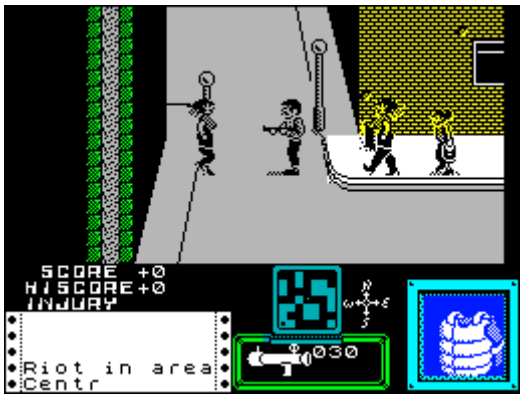
Краткое содержание картины. Скромный, хотя и талантливый архитектор, враг всякого насилия, дрожащими руками берется за оружие в первой серии картины, когда грязные бандиты убивают его жену и тяжко травмируют дочь. Выйдя в одиночку на ночные улицы города, он начинает свою большую и страшную месть всем, кто не дает людям спокойно жить. Постепенно в его действиях начинает проявляться профессионализм. Пролив реки крови, он остается на свободе, поскольку полиция, мягко говоря, смотрит сквозь пальцы на его подвиги - во всяком случае работы ей становится с каждым днем все меньше и меньше.

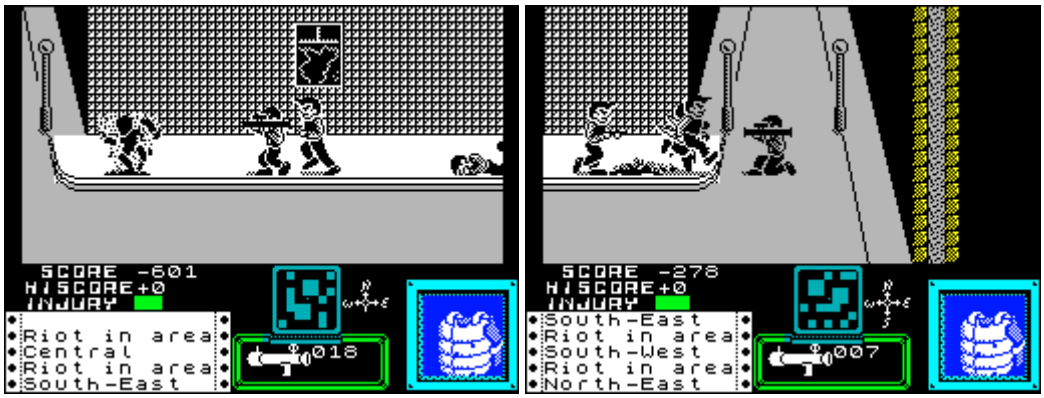
К третьей серии картины он уже сделал несколько безуспешных попыток "завязать", но обстоятельства по-прежнему вынуждают его снова и снова браться за оружие, Теперь он уже большой профессионал с известным по всей стране прозвищем "Мститель". Несколько разрушенных до основания кварталов и полное очищение города от неуправляемых банд - достижение третьей, но не последней серии картины.

Игра относится к жанру ACTION, но поскольку по ходу игры необходимо собирать оружие, и есть возможность произвольного выбора маршрута, следовательно в ней есть элементы ARCADE/ADVENTURE.

Сюжет: полиция не может освободить город от бандитов, ведущих между собой борьбу за власть в городе. Вооруженные банды устраивают стычки с полицией, во время которых проливается кровь мирных жителей.

Вы в качестве главного героя должны навести порядок в городе, а так как Вы свободны и независимы, в отличие от полиции, которая связана законами и уставами, то можете действовать более решительно и смело.





После загрузки программы нажмите "BREAK". Вы увидите нашего главного героя с винтовкой - это Стив, которым Вы управляете. Теперь можно начинать игру.

В нижней части экрана выводятся:

- выбранный вид оружия и боезапас;
- информационное табло, где приблизительно указывается местонахождение главарей банд (их пятеро и находятся они в пяти районах города: NORTH - WEST, NORTH - EAST, SOUTH - WEST, SOUTH - EAST, CENTRAL);
- состояние бронежилета (повреждения от выстрелов, возможность замены на новый);
- карта участка города показывает ваше местоположение (с указанием направлений: север, юг, запад, восток. Синий цвет карты - Вы настроены на поиск оружия, желтый - поиск главарей банд);
- SCORE - заработанные Вами очки;
- HI SCORE - высший результат предыдущих игр;
- INJURY - состояние Вашего здоровья (если показалась зеленая полоса, спрячьтесь в укромном месте и отдыхайте до тех пор пока полоска не исчезнет);

Итак начинайте игру. На улицах и в домах на Вас нападают бандиты, вооруженные винтовками или дубинками. Они не церемонятся - нападают и убивают на месте. Иногда будут пробегать полицейские и стрелять по бандитам, но от них мало пользы и рассчитывать Вам надо только на свои силы.

Оружие. У Вас имеется:

- пистолет;
- автомат;
- карабин;
- бронежилет.

Оружие и бронежилеты находятся в разных частях города - в домах, куда Вы можете зайти и пополнить боезапас или сменить вышедший из строя бронежилет.

Смена оружия производится нажатием клавиши "C". Нажатие клавиши "M" поможет Вам найти главарей банд, повторное нажатие "M" поможет найти оружие, боеприпасы, бронежилет. Пауза - клавиша "H".

Главари банд также находятся в домах. Для входа в дверь служит клавиша ENTER. Когда Вы найдете главаря, его надо расстрелять. Первым выстрелом его не убить, но зрелище того, как рассыпается стол, за которым он сидит, доставит Вам большое удовольствие.

Крайне нежелательно стрелять в простых мирных граждан - за это Вас штрафуют, но самое большее преступление, которое Вы можете совершить - убийство полицейского. За ним последует солидный штраф в очках и необходимость спасаться не только от бандитов, но и от полиции, а это уже конец.

FORUM

Проблемы ELITE

Мы по-прежнему получаем сотни писем по программе ELITE, но сейчас информация, содержащаяся в них, как правило повторяется. Какие темы в основном волнуют читателей:

1. Беспричинные захваты корабля на пиратских станциях. Очень много жалоб по этому поводу. После приземления сообщают, что Ваш корабль захвачен пиратами и действуют они безжалостно. Вам приходится начинать игру сначала. В качестве конкретного адреса приведем сообщение Тихомирова В. В. и Митрохина В. А. из г. Черноморска (Крым) о том, что так ведет себя станция на планете LAZASO в галактике N5.

2. По-прежнему много сообщений о чудачествах в версии, взломанной Родионовым. Здесь и галактика №47 и галактика № 1 без планеты LAVE и головокружительные суммы кредов и необъятные возможности вооружения - в общем, не знаем, хочется ли Вам в этот бред играть, но писать о нем нам не хочется.

3. После того, как мы в 10-м номере прошлого года опубликовали изображения всех документированных кораблей (как оказалось не зря!) пилоты начали настоящую охоту за НЛО. Сообщений много и как правило они имеют одиночный характер, но в десятках писем сообщается одно и то же. Есть корабль больших размеров, бесформенный, похожий на астероид, несущий на себе "Крейты" и "Саидуиндеры". Сам первым не нападает и, выпустив истребители, старается уйти от боя. Если его поразить, сбрасывает контейнеры. Факт существования такого недокументированного корабля можно считать установленным точно. Его поведение описывают десятки людей одними и теми же словами. Это кандидат на то, чтобы считаться "Космической платформой". Условно назовем его пока STRANGER ("странник").

Рисунок взят из письма Н. Ушакова (г.Ангарск) - см. рис.1.

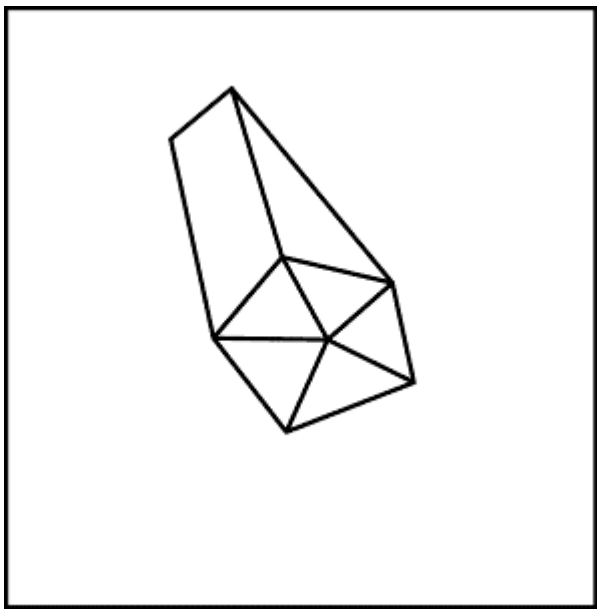


Рис.1

4. Есть разночтения по поводу подзарядки топливом от звезды. Мы не зря поднимали эту кажущуюся тривиальной проблему, 90% делают это не задумываясь, а 10% не могут сделать этого вообще. У них перегревается корабль задолго до включения надписи "Fuel scoops on" или просто температура не растет, но и заряда нет.

Может быть стоит прислушаться к мнению Михайлова С.Ю. (г. Саратов), который пишет, что "Cobra MK III" имеет нерегулярную геометрическую форму и поэтому есть разница, каким боком Вы ориентированы к звезде. Корабль, повернутый плоскостью, нагревается гораздо сильнее, чем корабль, развернутый торцом. "

Чтобы окончательно решить этот вопрос, пусть нам напишут только те, у кого заправка

никак не идет и сообщат о том, какой версией игры они пользуются. Может быть выяснится, что она у всех одна и та же.

В последнем выпуске (11-12) прошлого года мы сообщили о том, что пилоты начали исследования 102-байтного блока состояния, который отгружается, когда Вы сохраняете программу на ленте. Сегодня Радзевич А. А. из г. Нальчика дает дополнения к ранее неустановленным байтам.

15 - Ваш рейтинг.
16...18 - выполненные миссии.
41 - грузоподъемность (тонн).
61...66 - расположение звезд в галактике.
67 - ?
68 - координата у корабля.
69 - ?
70 - координата х корабля.

Интересное исследование провел пилот класса ELITE из г. Екатеринбурга Д. Палтусов.

Во-первых, он принудил свою версию программы работать с джойстиком, хоть она и очень не хотела:

```
POKE 29646,191: POKE 29649,226: POKE 29056,0: POKE 29659,0
```

(у него версия, помеченная Джеком О' Лантерном).

Во-вторых, он изучил машинный код программы и ему удалось установить кое-что интересное. Как оказалось, программа ведет внутри себя собственный подсчет очков.

За каждые 256 очков выдается сообщение RIGHT ON COMMANDER.

А вот как оценивается Ваша боевая активность при уничтожении кораблей:

THARGON - 4 балла.
FER-DE-LANCE - 3 балла.
ASP MK II - 3 Балла
PYTHON, ADDER-2, COBRA MK III, SIDEWINDER и "отшельник" - по 1 баллу.
Контейнер, астероид и VIPER - не дают очков.

Установлено соответствие и между набранной Вами суммой баллов и Вашим боевым рейтингом:

РЕЙТИНГ	ОЧКИ
HARMLESS	0
MOSTLY HARMLESS	9
POOR	17
AVERAGE	33
ABOVE AVERAGE	65
COMPETENT	129
DANGEROUS	768
DEADLY	2816
ELITE	6656

Кстати, это ответ тем читателям, которые недоумевают почему у них останавливается наращивание рейтинга после достижения COMPETENT.

Исследовав карты галактик, наш читатель обнаружил и невидимые звезды. Как и предполагалось ранее, это двойные звезды, получаемые наложением по OVER 1. Автору удалось просчитать их количество и расположение.

ГАЛАКТИКА	ЗВЕЗДЫ
2	DIMAATMA - ZAXERICE
4	QUZAARAR - RIUSBEQU
5	EDXEBERE - GEDIESQU
	LAZAZO - ZAENZA
	TEZABI - DIVEES
	ORLAED - TETIRI
7	DICELASO - ZAANXEGE

ERVEAN - SOINSOAN

8 CECEES - ESUSALE

Есть еще одна пара в 8-ой галактике, но найти ее пока нашему корреспонденту не удалось.

Итоги конкурса на лучший технологически развитый маршрут.

Читатели 1991 года помнят, что в N 10 "ZX-РЕВЮ" мы предложили пилотам игры "ELITE" составить для любой галактики план наиболее технологически развитого маршрута из 20 пунктов. Сумма технологических уровней в нем должна быть максимальной, сегодня мы можем подвести итоги и объявить пять победителей.

Всего на конкурс поступило около 30 маршрутов, как правило, пилоты, приславшие их, имеют рейтинг "DEADLY" ИЛИ "ELITE". Да это и не удивительно, маловероятно, чтобы начинающий пилот смог облететь восемь галактик и найти наиболее интересные с этой точки зрения планеты.

Как оказалось, удача в поисках сопутствовала тем, кто удачно определился в выборе галактики для прокладки своего маршрута. Наиболее широко были представлены галактики N1, 2, 3, 7, 8 и вот каковы средние результаты в этих галактиках:

N1 - 198
N2 - 219
N3 - 206
N7 - 240
N8 - 219

Нет ничего удивительного, что те пилоты, которые выбрали для исследования галактику №7 оказались в наиболее выигрышном положении.

Вот первые 5 маршрутов-победителей (все они проложены в седьмой галактике).

1.

Пилот: Марченко А.
Порт приписки: г. Новая Каховка (Херсонской области).
Боевой рейтинг: "ELITE"
Боевой стаж: 2.5 мес.
Маршрут:
BIONBIIN - MARAUS - CERIANON - ENARINES - ESTIRI - ONANORRA - XEUSREOR - MARARERE - ZARAUSXE -AGEBI - ESZAUSVE - ANGERIRI - QUGEZA - ESDITI - DIUSACE - ESRIXEAR - TIREGEES - ORESATRA - QUDIOR - ATESLETE.
Суммарный технологический уровень маршрута 251 балл.

2.

Пилот: Минеев А. В.
Порт приписки: Владивосток
Боевой рейтинг: not supplied
Боевой стаж: not supplied
Маршрут:
BIONBIIN - MARAUS - CERIANON - ENARINES - ESTIRI - ONANORRA - XEUSREOR - MARARERE - ZARAUSXE - XEVEXEAN - AGEBI - ESZAUSVE - QULAAON - ANGERIRI - QUGEZA - ESDITI - AEDEDLE - DIUSACE -TIREGEES - ESRIXEAR
Суммарный технологический уровень маршрута - 248 баллов.

3.

Пилот: Ветлянский А. В.
Порт приписки: Анадырь
Боевой рейтинг: not supplied
Боевой стаж: not supplied
Маршрут:

BIONBIIN - MARAUS - CERIANON - ENARINES - ESTIRI - ONANORRA - XEUSREOR - MARARERE - USQUEM - ZARAUSXE - AGEBI - ESZAUSVE - ANGERIRI - QULAAON - ISARE - ESDITI - DIUSACE - AEDEDLE - ESRIXEAR - TIREGEES

Суммарный технологический уровень маршрута - 248 баллов.

4.

Пилот: Сытник В. А.

Порт приписки: г. Угледар, Донецкой обл.

Боевой рейтинг: not supplied

Боевой стаж: 3.5 года

Маршрут:

BIONBIIN - MARAUS - CERIASON - ENARINES - ONANORRA - XEUSREOR - MARARERE - USQUEN - ZARAUSXE - AGEBI - ISARE - QULAAON -ESZAUSVE - ANGERIRI - QUGEZA - ESDITI - DIUSACE - AEDEDLE - ESRIXEAR - TJREGEES

Суммарный технологический уровень маршрута - 246 баллов.

5.

Пилот: Старцев А. Г.

Порт приписки: г. Миасс, Челябинской обл.

Маршрут:

TEABI - ERENRA - XEGEARER - BIONBIIN - MARAUS - CERIANON - ENARINES - ESTIRI - ONANORRA - XEUSREOR - MARARERE - USQUEN - ZARAUSXE - ISENES - ISARE - QUGEZA - QULAAON - AGEBI - ANGERIRI - ESZAUSVE

Суммарный технологический уровень маршрута - 242 балла.

Как Вы можете убедиться, все маршруты-победители имеют немало общих пунктов. Таким образом, можно считать, что наиболее технологически развитая область в программе ELITE установлена с достаточной точностью.

В соответствии с условиями нашего конкурса все пилоты прислали вместе с маршрутами выписки из бортжурнала, в которых указано какие товары целесообразно покупать на каждой из планет, и вот, что мы установили:

Посетив 100 планет, участники конкурса 29 раз проголосовали за приобретение компьютеров и 19 раз - за меха. Прочие товары уступают по своей эффективности. Вывод очень прост: хотите иметь меха - осваивайте компьютеры. Эта мысль может иметь далеко идущие последствия.

Победителям мы высылаем ксерокопию повести "THE DARK WHEEL", написанной по мотивам программы "ELITE".

Теперь новое предложение:

Мы предлагаем проложить наиболее рискованный маршрут. Условия те же. В любой галактике облететь 20 планет, не побывав на одной планете дважды и подсчитать уровень опасности этого маршрута, давайте условимся, что за анархическую планету начисляется 5 баллов, за феодальную - 4 балла, а за любую другую - 1 балл.

Точно так же, как и в прошлый раз, просим прислать выписку из бортжурнала, в которой указано, какие товары наиболее целесообразно покупать в каждой из точек вашего маршрута, чтобы иметь в очередном пункте максимальную прибыль. Эти данные будут свидетельством того, что Вы действительно пролетели по маршруту, а не взяли его из Всегалактического справочника.

Кроме того, интересно узнать, а на чем же делают бизнес в самых злачных уголках вселенной.

Задача несравненно более трудная, по сравнение с той, что была поставлена в прошлый раз. Мы знаем, что у этих планет Вас будут подстерегать пиратские флотилии, но зато и ценность ваших наблюдений будет максимальной. Начинающим пилотам можно будет выдать рекомендации о том, в какие районы им не следует показываться, а опытный

бойцам высокого класса будет ясно, где наиболее быстро можно поднять свой рейтинг.

Для документальности Ваших отчетов Вы можете сообщить свой боевой рейтинг и стаж полетов. Будет приятно также напечатать о количестве одержанных Вами побед во время прохождения маршрута.

Призы те же - 5 ксерокопий повести "DARK WHEEL" и публикация отчетов в майском/июньском выпуске.

Отчеты принимается до 1.05.92 по почтовому штемпелю даты отправки.

POKES

Д. Палтусов, который только что поделился своими результатами исследования ELITE активно исследует и другие программы. Вот список некоторых POKES, которые ему удалось отыскать самостоятельно:

TUTANKHAMUN	27783,0
MOTOS	48241,0
LODERUNNER	35427,24: 35426,0
DOWN TO EARTH	40142,195
SANXION	36584,0
KNIGHT LORE	53567,0: 50205,0: 50206,0: 50207,0:
SPELLBOUND	27038,8 (в начале)
BATTY	48437,167
CHRONOS	53407,N
BOULDER 4	30960,N
COMMANDO	27652-27657,0
BOULDER 1	31007,0: 31008,0: 31009,0
METAL ARMY	48535,0: 48559,0 (энергия) 42198,0: 48700, 107 (жизни)
KRAKOUT	46565,0: 61534,0
N.O.M.A.D.	40167,0
RENEGADE	41047,36
ACADEMY	31378,N: 31386,N 31249,N: 31305,N (N-оснастка корабля)
EQUINOX	41914,0
HYSTERIA	44588,201
L. NINJA 2	36579,175:36578,0
GUNFRIGHT	49233,54: 49234,1: 49235,0: 49236,0
ASTEMEX	43516,0
AIR FORCE II	51904,0
CIBERNOID	39402,0
CIBERNOIDD 2	36197,0
FRED	31171,0
IMPACT	54500,183

Два слова о том, как ищутся адреса для POKES на примере программы N.O.M.A.D

Зная, что в программе в исходном состоянии игрок имеет 3 попытки, наш читатель предположил, что где-то есть ячейка памяти, в которой хранится это число. Можно также предположить, что засылается оно туда в начале работы программы через регистр A. Это, конечно не единственный способ засылки числа в память, но он достаточно удобен и широко распространен.

Просмотрев с помощью дисассемблера программу, он сразу отбросил те области, в которых хранится графика, спрайты, тексты и т.п., сосредоточившись на машинном коде. (Как можно всего за несколько минут прикинуть где что в программе находится, "ИНФОРКОМ" писал во 2-ом томе трехтомника по программированию в машинном коде.)

Такой беглый просмотр позволяет выделить для более подробного исследования область длиной всего в несколько Кб. В этой области ищется команда Ассемблера LD A,3. Это занимает еще несколько минут. Если таких точек несколько, их надо будет все испытать, что делается следующим образом.

Рассмотрим фрагмент программы:

```
.............
LD A,3
LD (34586),A
LD (34480),A
LD HL,27304
LD (34584),HL
.............
```

Итак, есть подозрение, что в ячейке 34536 или 34480 организована переменная, в которой хранится количество попыток.

Давайте проверим ячейку 34586.

С помощью поисковых возможностей ДИСАССЕМБЛЕРа найдем где еще упоминается этот адрес и посмотрим, что там записано. Обычно это выглядит так:

```
.............
LD A,(34586)
DEC A
LD (34586),A
.............
```

Команда DEC A, уменьшающая значение в аккумуляторе на единицу, - прекрасный кандидат на то, что мы ищем. Ведь после гибели героя количество попыток уменьшается на одну.

Проверим это. Заменим DEC A на команду NOP (нет операции - ее код =0), и запустим программу после этой переделки. Если мы попали правильно, Ваш герой теперь бессмертен.

А вот как ищутся пароли для игры. Возьмем программу SABOTAGE, пройдя первый уровень удалось установить пароль второго (его программа выдала сама) - "BUMBLE BEE 2".

Теперь найдем место в программе, где хранится эта фраза, можно даже не пользоваться ДИСАССЕМБЛЕРом, а сделать это из БЕЙСИКа.

Код буквы "B" равен 66, а код буквы "U" - равен 85.

```
10 FOR i=25000 TO 65535:
      IF PEEK i = 66 AND PEEK (i+1) = 85
      THEN PRINT i
20 NEXT i
```

Когда будет найдено место в программе, в котором имеется сообщение "BU... ", компьютер выдаст вам его адрес, проверьте близлежащие адреса. В них может быть тоже что-либо интересное, так удалось установить пароли прочих уровней:

2) BUMBLE BEE 2.
3) HONORARIUM.
4) PHENOMENON.
5) ONOMASTICS.
6) SALMAGUNDI.
7) PSEUDONIMOUS.
8) ONOMATOPEDIA.

Мы благодарим Д. Палтусова за интересную информацию и вынуждены только пожалеть, что в ELITE COMPETITION он не занял призового места. Он нашел великолепный технологически развитый маршрут для галактики N1, но не догадался исследовать галактику N7, как это сделали победители.

Кстати, о паролях. Наш читатель Жукович Н. Н. из г. Усолье-Сибирское очень просит помочь ему найти пароль 8-го уровня в игре IMPACT. Может быть кому-то удалось его пройти? Для тех же, кто хочет попробовать свои силы в этой увлекательной игре, он сообщает пароли прочих уровней.

В игре 9 уровней по 10 экранов:

Экран 1 - пароль не требуется,

Экран 11 - EGGS
Экран 21 - CHIP
Экран 31 - LEAD
Экран 41 - TICK
Экран 51 - CASE
Экран 61 - FACE
Экран 71 ????
Экран 81 - USER - пользовательский.

Наш читатель из г. Смоленска, Коновалов А. В. тоже активно работает с POKES. К сожалению, он отмечает, что некоторые POKES из тех, что были опубликованы в №10 у него не пошли. В то же время, известно, что по стране ходят многочисленные версии даже для одной и той же игры. Поэтому для тех, у кого не пошли какие-то POKES, он предлагает воспользоваться прилагаемым ниже набором.

Все прилагаемые POKES вставляются после последнего LOAD"", а не между USR.

GAMEOVER1	39334,0 (жизни)
	32417,0 (гранаты)
HORACE AND SPIDERS	27680,60
MAG-MAX	58472,60
MUTANT MONTY	55761,60: 56483,183
CYBERNOID-1	39403,0
VIXEN-III	41423,0
FIREBIRDS	27235,0
XEVIOUS	53592,n
BLADE WARRIOR	37161,0
	39490, 60
KARNOV	25620,0
GAMEOVER 2	38704, 0 (жизни)
	32388,0 (гранаты)
GONZZALEZZ 2	35749,0 (жизни)
	48056,0 (оружие)
ASTEMEX	43517,52
GONZZALEZZ 1	37085,0
METAL ARMY	36697,0 (жизни)
	42650,0 (энергия)
ACTION FORCE 2	47874,182
KRION	45004,0 (жизни)
	44486,0 (горючее)
GLUG GLUG	34139,0
PSSST	24984,0
ROGUE TROOPER	30924,0
EQUINOX (для версии ROBY'86)	39858,52:39659,182 (жизни)
	48691,0:48762,0 (горючее и лазер)
STAINLESS STEEL	46957,60
UNDER WURLDE	59376,0
RICK DANGEROUS	55460,0 (жизнь)
	61045,0 (гранаты)
	60954,0 (патроны)
CAVELON	47250,0
R-TYPE	30873,0
DAN DARE 2/1	46459,0
VIGILANTE	48735,60
KOKOTONI WILF	43742,0
PETER PACK-RAT	27243,100

PENTAGRAM	49917,0
THUNDERBLADE	33199,0
	33145,0
THOR	37550,195
DAN DARE 2/2	51797,0
MOTOS	42241,0
CHICAGO	60264,0
FROST BYTE	30992,183 (жизнь)
	28237,183 (время)
RUFF & REDDY	33567,0
PROJECT FUTURE	29332,0
JETMAN	36965,0

Комментарий "ИНФОРКОМа".

Дорогие друзья! То, что игры имеющие хождение в стране, уже давно не являются первозданными, Вам хорошо известно. Сначала их курочат в Голландии, потом через ФРГ это поступает в Польшу, где дело поставлено не хуже, чем у нас, и только потом мутными ручейками программы поступают на расправу к нашим специалистам.

Вам это известно очень хорошо.

Кроме того, для различных POKES существует и разная технология их введения, что тоже немаловажно и не всегда правильно выполняется. Радикальных рецептов быть не может, кроме одного: учится, учится и учится, как завещал сэр Клайв Синклер, а в Англии, как Вы понимаете, звание сэра и титул лорда простому инженеру за одни красивые глаза не дают.

Осваивайте машинный код, изучайте опыт и приемы тех, кто это уже освоил, развивайте собственные приемы. В принципе, на страницах "ZX-РЕВЮ" мы даем достаточно информации, чтобы было с чего начать. Как Вы увидите, в этом году в "ZX-РЕВЮ-92", будет немного больше материалов для тех, кто осваивает машинный код.

Только освоив технологию Вы почувствуете себя хозяином положения и сможете воспользоваться чьим-то POKE с полным знанием дела или адаптировать его под свою конкретную программу.

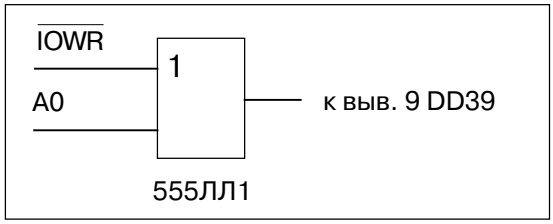
Вопросы совместимости.

Мы продолжаем рассматривать вопросы совместимости отечественных моделей ПК с фирменным программным обеспечением. Наши читатели уже встречались с разработками в этом направлении, ведущимися Полубарьевым Сергеем Викторовичем из г. Йошкар-Ола.

Сегодня речь идет о наведении порядка с дешифрацией портов ввода/вывода в версии Зонова (15 корпусный "Ленинград-1") для обеспечения полной совместимости с оригиналом.

Вот последовательность доработки. Для этого необходимо установить на свободном месте платы по одной микросхеме К555ЛЛ1 и К555ЛИ1 и далее выполнить в схеме следующие изменения:

1) Отключить сигнал IOWR от вывода 9 DD39 (К555ТМ9 все обозначения по принципиальной схеме "Ленинград-1") и собрать на одном из элементов К555ЛЛ1 следующую схему:



После этого запись в порт вывода будет происходить только по четным адресам, что исключает мерцание и самопроизвольное изменение цвета бордюра.

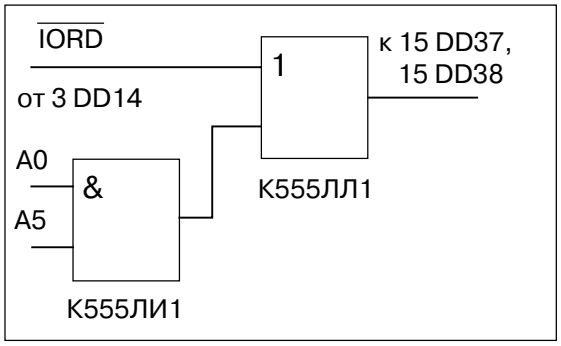
Небольшое отступление перед последующими доработками:

"Ленинград-1", по-видимому, разрабатывался под самодельный джойстик с нормально-замкнутыми контактами. Наиболее же распространенными являются джойстики "Atari"-типа, у которых контакты нормально разомкнуты и замыкаются при отклонении рукоятки. Часть последующих переделок будет направлена на обеспечение совместимости именно с ними.

2) Резисторы R21...R25 сопротивлением 15 кОм, подключенные к линиям джойстика DV0...DV4 соответственно, заменить на МЛТ-0,125 с сопротивлением порядка 660 Ом...1.5 кОм. Общую точку их соединения отключить от шины "+5V" и соединить ее с "землей".

3) Свободные входы DD37 (выводы 6,10,13) соединить с выводом 8 этой же микросхемы.

После выполнения этих доработок в принципе уже можно включить компьютер и оценить результаты. Команда PRINT IN 31 должна при отпущенном джойстике печатать 0, а при отклонении рукоятки или нажатии кнопки - некоторое число, зависящее от положения джойстика. Запись любого числа по любому нечетному адресу OUT не должна изменить цвет бордюра. Однако, если Вы хотите довести джойстик до полного Кемпстон-стандарта, а заодно и освободить часть нечетных портов ввода, необходимо сделать еще одну доработку:



4) Отсоедините сигнал IORD, поступающий с вывода 3 DD14 (К555ЛЛ1) от вывода 15 DD37 и DD38, и соберите следующую схему его дополнительной доработки:

Теперь считывание из портов будет производится так:

A0=0 и A5=1 - клавиатура;

A0=0 и A5=0 - клавиатура; т.е. клавиатура читается по любому четному адресу порта.

A0=1 и A5=0 - Кемпстон-дж-к.

A0=1 и A5=1 - несуществующий порт, т. е. все нечетные порты с адресами, большими, чем 31 свободны.

Практика показывает, что при выполнении доработок 1...3 большинство проблем исчезает. Доработка 4 не является обязательной (в игровых программах обычно эффекта не дает) и применяется во-первых, чтобы обеспечить соответствие стандарту и, во-вторых, чтобы освободить часть адресного пространства, например для установки "ZX-LPRINT III".

Соответственно, если необходимости в ней нет, микросхему К555ЛИ1 можно не устанавливать, оставшиеся свободными элементы дополнительных ИМС можно использовать, например, при подстыковке интерфейса "BETA-DISK".

Если рассмотреть доработку, предлагавшуюся в ZX-РЕВЮ в прошлом году (стр. 157, рис. 7), то можно понять, что в принципе она делает то же самое, но при этом со старших 3 битов порта 31 продолжает считываться единица, что в принципе может в некоторых случаях приводить к нарушению совместимости и в некоторых играх и приводит.

Теперь несколько слов об отмечавшемся замедлении работы игровых программ на "Ленинграде-1" (см. ZX-РЕВЮ-91. с. 197).

Прежде всего, примем за аксиому следующее:

а) Схема построена так, чтобы использовать кварцы от 13.0 до 14.5 МГц. Тактовая частота процессора получается при делении частоты генератора на 4, т.е. это 3.5 МГц при частоте кварца 14 МГц. Поэтому первым делом проверьте кварц, который стоит в Вашей машине.

б) Конфликт дисплея и процессора (о котором писали на странице 52 "ZX-РЕВЮ-91") может происходить по любому адресу ОЗУ, поскольку у "Ленинграда-1" сплошное поле памяти 48К. В этом случае процессор приостанавливается по сигналу WAIT, формируемому на триггере D 9.2. Исходными для формирования служат сигналы M1 процессора и CSRAM (выборка ОЗУ поступает на вывод 1 регистра К555ИР22). Однако, в некоторых схемах и рекомендациях по наладке (соответственно и в платах) вместо сигнала CSRAM требуют использовать сигнал MREQ. Последствия этого - замедление работы даже с ПЗУ (хотя на некоторых экземплярах компьютера это, возможно, и необходимо).

Методы устранения несложны.

а) Замените кварц на 14,0 или 14,5 Мгц. Чтобы при этом не нарушалась синхронизация TV, внесите изменения в схему включения счетчика D4 (К555ИЕ7) согласно следующей таблицы:

Частота, МГц	Выводы D4 соединить с:	
	"+5 вольт"	"общий"
13,0	10, 15	1,9,14
13,5	10	1,9,14,15
14,0	1, 15	10,9,14
4,5	1	10,9,14,15

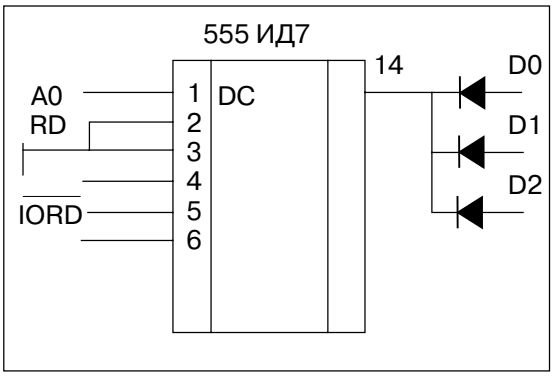
Если после замены кварца обнаружатся сбои ОЗУ, придется поменять микросхемы на более быстродействующие или попытаться подстроить временные диаграммы сигналов RAS и CAS.

6) Решение второй проблемы вытекает из самой проблемы, однако, если после замены на формирователе MREQ на CSRAM обнаружатся сбои ОЗУ, придется вернуть все на место.

В любом случае рекомендуется после проведения доработок выполнить длительную серию тестов ОЗУ, чтобы убедиться в надежной работе компьютера.

Все предлагаемые доработки проверены автором лично на практике, во избежание появления сбоев лучше применять в компьютере "Ленинград-1" только серию К555 (К1533), не заменяя ее на К531 и К155 - тогда наладка почти не требуется.

Для тех, кто работает с Краснодарским и Харьковским вариантами компьютера, впрочем как и с "Ленинград-1", наш читатель из Харькова В.И. Бойко предлагает несложную доработку, обеспечивающую работу таких программ, как "ARKANOID", "DUET" и некоторых других.



На ножку 6 микросхемы подается старший байт адреса, формируемый схемой генерации адреса ULA. Все остальные сигналы можно взять непосредственно с процессора.

Просьба о помощи.

В Межшкольном комбинате г. Майли-Сай (Кыргызстан) установлена локальная сеть на базе "Спектрумов", изготовленных в Ташкенте. Проблема с эксплуатацией программ в сети. Программы есть, но по сети не идут. Если у кого есть опыт эксплуатации сети, просьба отозваться. "ИНФОРКОМ" рассматривает вопросы обеспечения работы школ, как самые приоритетные.

Контактный адрес: 715420, Кыргызстан, Джалал-Абадская область, г. Майли-Сай, ул.

Маяковского. 2, кв. 21, Курику В. Н.

Сам "ИНФОРКОМ" располагает всей необходимой информацией по тому, как организуется локальная сеть для фирменных машин на базе интерфейса локальной сети ZX-INTERFACE I. Известна система команд, системные переменные, коды-перехвата, но мы совершенно не знаем, на каких интерфейсах делают сети у нас и какими командами они задействуются.

Примем к печати статьи на данную тему.

И снова SHERLOCK.

Нашим читателям уже известны проработки эксперта Ескевича А. А. из г. Новосибирска. Как студента-филолога, его конечно интересуют текстовые игры (жанр adventure) и он активно взялся за работу с программой SHERLOCK, но столкнулся с проблемой.

Узнав от Уотсона о том, что в Лизерхэде (LEATHERHEAD) произошло убийство двух человек, он решил направить Холмса на место преступления. Проблема в том, что ни один извозчик не желает его понимать. Он побывал на всех четырех вокзалах и на прилегающих к ним улицам (Aldereate street, Bishops Road, Backingham Palace и King Cross), но ни один извозчик ниоткуда не желает везти его в Лизерхэд.

Мы с большим удовольствием принимаем эту жалобу на безобразную работу лондонских извозчиков и должны заметить, что у нас в Москве тоже очень часто бывает, что таксисты не хотят Вас везти ниоткуда и никуда. Не знаем, как обстоят дела в Новосибирске. С другой стороны, есть сведения о том, что некоторым удается как-то договорится с водителем, возможно, есть какие-то филологические тонкости в общении пассажира и водителя нам неведомые.

Попробуйте применить такой же подход и к Лондонским кэбби. Одним словом: "Вам нужно найти необходимые слова". Вот они:

Выйдя из дома, Холмс спешит на утренний поезд в Лизерхэд, который отправляется в 9:15. Для этого ему нужно нанять кэб:

1. HAIL CAB
2. CLIMB INTO CAB
3. SAY TO CABBY "GO TO..."

Вот и все. Этим же методом Вы будете нанимать кэб и в других подобных ситуациях.

На платформе Холмс встретит инспектора Лестрейда... а далее желаем Вам успеха.

А вот, что пишет Антон Скворцов из С. Петербурга. "Меня сразу заинтересовали ваши статьи об адвентюрных играх. Мне они во многом помогли. Правильно, что Вы развиваете интерес к этому жанру.

Я. правда, еще не очень опытен и хотел бы переписываться с кем-нибудь, кто любит эти игры. Проблема в том, что на нашем "толчке" их не достать, и я пока довольствуюсь двумя программами - "Knight-Tyme и The Hobbit. "

Уважаемый Антон, развивая это направление, мы сознательно шли против течения, против моды, против законов местных "толчков". Да, сегодня энтузиастов этого жанра пока мало. Но есть мировой опыт, и он гласит однозначно - через два-три года общения с компьютером основными становятся игры трех жанров - ADVENTURE, STRATEGY, MANAGMENT. Так что не беспокойтесь, если на Вашем "толчке" есть умные предприниматели, они это поймут, перестроятся и в течение года - двух ситуация резко изменится.

Наша задача - подготовить этот поворот, а Вы правильно делаете, что осваиваете это направление. Когда придет его время, будете уже иметь нужный боевой (а может быть и деловой) опыт.

Для любителей "умных" игр, желающих связаться с Антоном: 195027, С.-Петербург, Среднеохтинский проспект, д. 2 "А", кв. 52.

Внимание! Будьте осторожны.

Наш читатель из г. Харькова Хоминич Р. В. сообщает о неприятном моменте, который поджидает многих любителей аркадных адвентюр, начавших игру в программу BLACK RIDER фирмы TOPOSOFT,1988.

Прежде чем начать с ней работать, желательно проверить ее на полноту состава файлов.

А вот в чем суть.

Вы являетесь капитаном пиратского корабля, корабль находится на стоянке в бухте и ремонтируется. С каждым днем становится все труднее управлять разлагающейся командой, матросы пьют ром и готовы разбежаться.

Ваша первая задача - сохранить команду. Разрушить сходной трап можно выстрелом из пушки, но надо найти фитиль (факел).

Поиск ведется в многочисленных сундуках с сокровищами, расположенных в различных помещениях корабля. Чтобы не тратить время на возню с замками, капитан открывает их не церемонясь - выстрелом из пистолета.

Среди прочего барахла в этих сундуках Вы найдете и полезные для себя веши.

Все время Вам придется отбиваться от своих подгулявших подчиненных. Интересна система подкрепления энергии. Подкрепиться можно ромом, но будьте осторожны, не переберите.

Когда Вы обыщете все сундуки, на самой нижней палубе Вас будет ждать сюрприз, старый красный сундук, который до этого никак не открывался, вдруг окажется открытым, а рядом с ним лежит уникальная карта острова сокровищ. Вы бросаетесь к ней и… программа просит загрузить следующий уровень, что очень неприятно, если у Вас его нет, а именно в таком виде программа и нашла широкое распространение.

Тех счастливых джентльменов удачи, которые владеют полной игрой, мы просим прислать список ее файлов с указанием длины и названия, дабы многочисленные поклонники аркадных адвентюр не становились жертвами столь бездушного обращения.

"РЕГИСТРАТУРА" - система многоцелевого назначения.

Система относится к разряду информационно-поисковых и справочных систем, исполняемых и настраиваемых "под заказчика". Данная система может удовлетворить большинство потребностей организаций в удобном программном средстве для хранения и обработки текущей информации.

I. НАЗНАЧЕНИЕ СИСТЕМЫ

Система предназначена для ведения учета и проведения обработки любой информации, необходимой на Вашем предприятии (в Вашем учреждении).

Например:

- регистрация входящих и исходящих документов;
- регистрация входящих и исходящих товаров;
- регистрация пациентов в лечебном учреждении;
- регистрация клиентуры и заказчиков
- и многое, многое другое.

Система может удовлетворить потребности отдела кадров, планово-финансового отдела, отдела соцкультбыта, различных административных, муниципальных и хозяйственных служб и т.п., КРОМЕ БУХГАЛТЕРИИ.

II. ВОЗМОЖНОСТИ СИСТЕМЫ

Система позволяет:

- вводить информацию в любое заданное количество полей;
- просматривать информацию в избранном формате;
- вносить изменения;
- сортировать информацию по любому полю или по любой их совокупности;
- производить поиск по заданному критерию или по любой их совокупности;
- произвольно по каждому критерию задавать ключевое соотношение на поиск;
- получать статистическую информацию (производить подсчет записей, удовлетворяющих любым заданным критериям при любых заданных соотношениях);
- распечатывать всю или избранную информацию на принтере в избранном составе, избранным шрифтом на листах избранного формата;
- делить избранные файлы на части в автоматическом или ручном режиме;
- объединять файлы;
- …. и многое другое.

Практически все операции, включая выбор формата просмотра, выбор критериев на поиск и сортировку, выбор ключевых соотношений и т. д. автоматизированы и не вызывают трудностей у неподготовленного пользователя. Система имеет приятный дизайн, удобную, не вызывающую утомления систему управления 6-ю клавишами, доставляет радость в работе.

Основной принцип: пользователь должен только ввести информацию в первый раз - все остальное выполняется простейшим выбором из меню альтернативных предложений.

Несмотря на то, что освоение системы неподготовленным пользователем может проходить на интуитивном уровне, к ней прилагается краткая инструкция, гарантирующая освоение системы в считанные часы.

Эта система - наилучший ответ на Ваши самые срочные потребности. Хотите немедленно ощутить мощный эффект от внедрения ЭВМ в Вашу организацию - воспользуйтесь этой системой.

Введенная Вами информация не пропадет и при переходе в дальнейшем на другие, более мощные системы, в том числе и работающие в локальных сетях. Вся введенная Вами информация хранится в формате .DBF, который во всем мире признан неофициальным стандартом.

III. КОМПЛЕКТ ПОСТАВКИ

Система поставляется на одной дискете 5.25" (MS DOS, 360 K). На титульном экране системы проставляется название Вашей организации.

IV. ИСХОДНАЯ ИНФОРМАЦИЯ

Поскольку система исполняется персонально "под заказчика", от Вас необходимо получить исходную информацию - заверенный руководителем СПИСОК НЕОБХОДИМЫХ ВАМ ПОЛЕЙ ИНФОРМАЦИИ И ИХ РАЗМЕР, например:

1. ФАМИЛИЯ - 16 знаков
2. ИМЯ - 12 знаков
3. ОТЧЕСТВО - 16 знаков
4. ДАТА РОЖДЕНИЯ - 8 знаков
5. ДОМ. АДРЕС - 40 знаков
6. ПОЛ - 1 знак

.....

5. ПРИМЕЧАНИЕ - 20 знаков

Просим Вас:

1. Не задавать размер полей более 40 знаков. Например, поле адрес можно разделить на 3 поля ПОЧТ. ИНДЕКС, АДРЕС, ТЕЛЕФОН.

2. Не задавать названия полей длиннее 16 знаков. Пользуйтесь сокращениями. Например: ДАТА ОКОНЧ. ВУЗА.

V. СРОК ИСПОЛНЕНИЯ ЗАКАЗА.

Срок исполнения - 2 - 3 недели после поступления средств на наш р/с и заказа с заверенным СПИСКОМ ПОЛЕЙ ИНФОРМАЦИИ.

VI. ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ К АППАРАТНО-ПРОГРАММНОМУ ОКРУЖЕНИЮ

1. Полная аппаратно-программная совместимость с IBM PC XT/AT. Надежность функционирования на отечественных модификациях не гарантируется и не обсуждается.

2. Наличие "жесткого" диска ("Винчестера") стандартного объема.

3. Дисковод гибких дисков 5,25" или 3.5"". Эта система поставляется нами без защиты от копирования.

4. Операционная система - MS DOS не ниже 3.20.

5. Русификация компьютера в стандарте ГОСТ (кодировка альтернативная).

6. Требования к монитору - не специфицируются, желательно - EGA.

7. Требования к принтеру - совместимость со стандартом EPSON.

VII. ГАРАНТИЙНЫЕ ОБЯЗАТЕЛЬСТВА

Гарантийным свидетельством при поставке программного продукта является картонный альбом, в который вложены дискеты с указанной на нем датой продажи. При его отсутствии поставка выполняется с заверенным гарантийным талоном.

Гарантиями обеспечивается:

- бесплатная замена поставочных дисков, неработоспособных в состоянии поставки (в течение месяц после поставки);
- замена с минимальной оплатой при выходе программ из строя по вине пользователя (механическое или электромагнитное повреждение, поражение вирусом на машине пользователя и т. п.) или по истечении месяца после поставки. Минимальная оплата не превышает стоимости дисков + 5% текущей стоимости программного обеспечения + стоимость почтово-транспортных расходов и согласовывается с потребителем.

VIII. ПОРЯДОК ОФОРМЛЕНИЯ ЗАКАЗА.

а) Направить в наш адрес письмо заказ с указание необходимого программного продукта и количества копий. Приложить копию платежного поручения и заверенный СПИСОК ПОЛЕЙ ИНФОРМАЦИИ И ИХ РАЗМЕР.

Наш адрес: 107241, Москва, Б-241, а/я 37, "ИНФОРКОМ"

б) Произвести предварительную оплату платежный поручением на наш р/с: N 500461778 во Фрунзенском коммерческом банке г. Москвы.

Стоимость системы на период март-апрель 1992г. - 7200 рублей + 28%

"ЗЕЛЕНЫЙ ПАКЕТ" ДИСТРИБУТОРА

О том, что такое "зеленый пакет" Вы можете подробно прочесть в N11-12 "ZX-РЕВЮ" за 1991 г.

Стоимость "зеленого пакета" по системе "РЕГИСТРАТУРА" составляет для частных лиц:

Рабочая версия программы -	10% * 7200 = 720
Дискета -	100
Почтовые расходы-	5
Итого	825 рублей

Уточняем, что высокая стоимость дискеты вызвана тем, что нашим дистрибуторам поставляются дискеты особо высокого качества с защитным тефлоновым покрытием, имеющие особый представительский вид и высокую коммерческую стоимость.

Напоминаем, что затраты дистрибутора на "зеленый пакет" являются только залогом и возвращаются по требованию. Активно работающим дистрибуторам затраты возвращаются при выплате комиссионных и далее такие пакеты предоставляются бесплатно.

Содержание

СПЕКТРУМ В ШКОЛЕ 3

1. "LISTNAME" 3
2. "REMONT" 4

BETA BASIC 6

ГЛАВА 1. ВВЕДЕНИЕ 6
Особенности работы с языком БЕТА-БЕЙСИК 3.0 7
Работа с более ранними версиями БЕТА - БЕЙСИКа 7
Загрузка и выгрузка программ, написанных в БЕТА-БЕЙСИКе 3.0. 8
ГЛАВА 2. РЕДАКТИРОВАНИЕ 8
Курсор текущей строки 8
Команда EDIT <номер строки> 9
Переключение режимов курсора 9
Управление вводом ключевых слов 9
Ввод ключевых слов БЕТА-БЕЙСИКа 9
Проверка синтаксиса 9
Команда LIST FORMAT 10
Команда CSIZE 10
Команды JOIN <номера строк> и SPLIT. 10
Управляющий код "новая строка". 10
ГЛАВА 3. ПРОЦЕДУРЫ 10
Передача параметров в виде ссылки 13
Передача параметров списком 14
Рекурсия 15
Ошибки 15
ГЛАВА 4. СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ 16
ГЛАВА 5. ОБРАБОТКА ДАННЫХ 16
ГЛАВА 6. ГРАФИКА 16
ГЛАВА 7. СЕРВИСНЫЕ И ОТЛАДОЧНЫЕ ВОЗМОЖНОСТИ 17

ЗАЩИТА ПРОГРАММ 18

ЧАСТЬ 1 18
ГЛАВА 1. Исключение возможности остановки программ. 18
1.1. Общие рекомендации. 18
1.2 Подпрограмма обработки сообщений с кодами D, H, L. 19
1.3. Подпрограмма обработки сообщений об ошибке, кроме 0:OK; 8:END OF FILE; 9:STOP STATEMENT. 21
1.4. Метод защиты, основанный на совмещении Бейсика и программы в машинных кодах. 21
1.5. Оригинальный метод защиты, использующийся при загрузке машинных кодов. 23
1.6. Метод защиты, используемый в программе FIST III. 24
ГЛАВА 2. Методы защиты от листинга или как сделать текст программы нечитаемым. 24
2.1. Общие рекомендации. 25
2.2. Универсальная система защиты - метод нулевых строк. 26
2.3. Использование управляющих кодов. 28

40 ЛУЧШИХ ПРОЦЕДУР 34

РАЗДЕЛ А 34
1. ВВЕДЕНИЕ 34
Общие сведения о Бейсике и машинных кодах 34
2. ВНУТРЕННЯЯ СТРУКТУРА ZX SPECTRUM 36
Карта памяти 36
PEEK и POKE 37
Дисплейный файл 38
Атрибуты 40
Буфер принтера 40
Область программ на BASICe 40
Цифровой пятибайтный формат 42
Область переменных 43
Подпрограммы ПЗУ 43
3. МАШИННЫЙ ЯЗЫК Z80 44
Регистры Z80A 44
О системе команд процессора Z80. 47
РАЗДЕЛ В 48

4. ВВЕДЕНИЕ 48
Загрузчик машинного кода *49*
5. Подпрограммы сдвига. 52
5.1 Сдвиг атрибутов влево. *52*
5.2 Сдвиг атрибутов вправо. *52*
5.3 Сдвиг атрибутов вверх. *53*
5.4 Сдвиг атрибутов вниз. *54*
5.5 Сдвиг влево на один символ. *55*
5.6 Сдвиг вправо на один символ. *56*
5.7 Сдвиг вверх на один символ *57*

МАСТЕРФАЙЛ 09 ПОЛНАЯ РУСИФИКАЦИЯ. 59

ВСТУПЛЕНИЕ. 59
1. Использование символов UDG-графики. *59*
2. Использование дополнительного символьного набора. *61*
3. Перевод программы на русский язык. *66*

СОВЕТЫ ЭКСПЕРТОВ 68

STALINGRAD 68
Введение. *68*
Управление игрой *69*
Некоторые детали *71*
Уровни сложности игры *71*
Выгрузка файлов на ленту *71*
Замечания *72*
FLIGHT SIMULATION 72
Аспекты полета *72*
Приборная панель *73*
Управление самолетом *74*
Навигация *75*
STAR RAIDERS. 75
Враги *75*
Запуск игры *76*
Экран *76*
Управление. *77*
Энергия *77*
THE TRAIN 78
Краткий сюжет *78*
Настройка программы *78*
Экран *79*
Управление. *79*
Мост (BRIDGE) *80*
Станция (STATION) *80*
Развилка или перекресток (SWITCH) *80*
DEATH WISH 3 81

FORUM 83

Проблемы ELITE 83
Итоги конкурса на лучший технологически развитый маршрут. *85*
POKES 87
Вопросы совместимости. 90
Просьба о помощи. 92
И снова SHERLOCK. 93