«ИНФОРКОМ»

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР «ZX-SPECTRUM»

Элементарная графика

Москва — 1993

<< ИНФОРКОМ >>

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР

"ZX - SPECTRUM"

Элементарная графика

Москва - 1992

СОДЕРЖАНИЕ

ОБ ЭТОЙ КНИГЕ

Бытовые персональные компьютеры типа "ZX-Spectrum" (система Sinclair) - являются одними из самых популярнейших в нашей стране и это не случайно. Долгое время они были и самыми популярными в мире. Не в последнюю очередь это связано с их графическими возможностями. Достигнутые сравнительно простыми техническими средствами, эти возможности таковы, что во многих случаях вполне способны удовлетворить насущные потребности серьезного пользователя и, несмотря на наличие определенных ограничений, они отличаются сравнительной простотой доступа, что нередко делает этот скромный бытовой компьютер достойной альтернативой более дорогим, но и более сложным ПЭВМ.

Компьютерная графика - это та сфера программного обеспечения, которая требует повышенного быстродействия от компьютера и его операционной системы, а также наличия значительных ресурсов оперативной памяти. Надо сказать, что встроенный БЕЙСИК, с которым работает Ваш компьютер - далеко не лучшая среда с этих двух точек зрения и самые значительные достижения в области графики на "Спектруме", конечно, реализуются не в БЕЙСИКе, а программированием в машинном коде процессора Z-80, который является сердцем Вашего компьютера.

В то же время, мы прекрасно понимаем, что программирование в машинном коде еще не стало общедоступным, особенно для тех, кто недавно приобрел компьютер и делает первые шаги самообразования. Эта книга поможет Вам продвинуться вперед. Мы учитываем определенную психологическую сложность немедленного перехода всех на работу в машинном коде и придали этой книге специальную структуру, которая позволит удовлетворить как тех, кто имеет глубокий опыт работы с компьютером, так и тех, для кого эта книга - первое знакомство с машиной.

Особый раздел составляют конкретные инструментальные программные средства, приведенные в третьей части книги. Они явятся, с одной стороны, практикумом для тех, кто самостоятель-

но изучает материал, а с другой - дадут будущему пользователю необходимый набор программных инструментов, с помощью которых он сможет строить и оттачивать свои программы.

Как и все материалы "ИНФОРКОМа", эта книга сделана в качестве учебного пособия для самообразования. В этом смысле она является законченным и самообеспеченным изданием. Но мы продолжаем работу над вопросами компьютерной графики и в развитие темы в ближайшие месяцы выпускаем последующие тома серии:

т.2 - "Прикладная графика". В нем рассмотрены вопросы конкретного применения графики для деловых, обучающих, научных, игровых и прочих программ. Рассмотрена растровая, векторная, блочная, трехмерная, теневая графика и пр.

т.3 - "Динамическая графика" - рассмотрены вопросы анимации графических изображений. Основная направленность - на создание обучающих и игровых программ.

т.4. - "Дизайн Ваших программ" - квинтэссенция того, к чему читатель шел, работая над первыми томами серии.

Таким образом, предложенная Вашему вниманию "Элементарная графика" является не только и не столько самостоятельным учебным пособием, но также и методологической базой для последующих изданий. Мы рассчитываем на то, что те, кому понравится эта книга, приобретут и последующие тома.

Облегчить работу с данной книгой Вам поможет наше другое издание "Персональный компьютер ZX-SPECTRUM. Программирование в машинных кодах". Два года назад оно выпускалось в трех томах и десятки тысяч любителей этого компьютера уже смогли с его помощью приобщиться к профессиональной работе с компьютером. Сейчас мы подготовили, выпустили и рассылаем заказчикам новое расширенное и дополненное издание, содержащее все три части в одном томе (271 стр.).

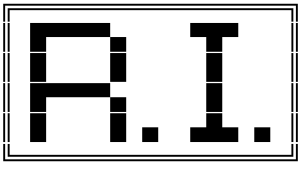
Часть 1 - "Первые шаги в машинных кодах".

Часть 2 - "Практикум по программированию в машинных кодах".
Часть 3 - "Справочник по программированию в машинных кодах".

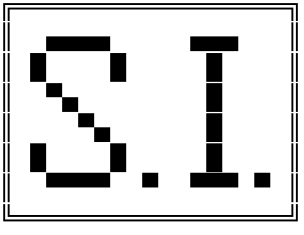
Мы ждем Ваши отзывы и пожелания, а также заявки от организаций, способных наладить печать и распространение в своих регионах наших изданий по оригинал-макету (поставляется на дискетах MS DOS для IBM-совместимых ПЭВМ). С предложениями обращаться по адресу:

121019, Москва, Г-19, а/я 16, "ИНФОРКОМ".

Postscriptum



Время от времени в нашей книге Вам будут попадаться такие марки R.I. Это наш сигнал RARE INFORMATION - РЕДКАЯ ИНФОРМАЦИЯ. Вам знакомо ощущение: "Где-то я это видел, но не помню где"? - так вот, когда Вы впоследствии будете листать эту книгу в поисках чего-то очень нужного, эта марка Вам поможет.



S.I. - SYSTEM INFORMATION - СИСТЕМНАЯ ИНФОРМАЦИЯ - чисто справочная информация по системе. Более чем вероятно, что обращаться к ней Вы будете еще не раз и не два.

I. ЭЛЕМЕНТАРНАЯ ГРАФИКА НА БЕЙСИКЕ

1.1. ВВЕДЕНИЕ

Возможно, что Вы сейчас сидите перед экраном своего монитора (телевизора) и пытаетесь найти ответ на вопрос - как же так происходит, что компьютер способен создавать на экране удивительные многоцветные изображения, с которыми мы нередко сталкиваемся в игровых программах и что необходимо знать для того, чтобы сделать это самому?

Ответ, как это ни парадоксально, на самом деле прост. Вы ведь видите, что изображение на экране состоит из элементарных точек, а значит имеет какую-то структуру и, очевидно знаете, что компьютер имеет оперативную память, состоящую из десятков тысяч ячеек. Поэтому нужно только знать, как связана структура того экрана, который Вы видите перед собой, с ячейками оперативной памяти и научиться управлять изображением, засылая в эти ячейки те или иные числа или подавая те или иные команды. Изучением этих вопросов мы и займемся на страницах данной книги, но сначала остановимся на некоторых особенностях "Синклер"-совместимых компьютеров.

Характерной особенностью этих машин является то, что экран одновременно является и символьным и графическим. Вы можете одновременно на экране и напечатать что-то оператором PRINT и нарисовать что-то, например оператором PLOT.

Сравните PRINT "." и PLOT 10,150. И в том и в другом случае на экране появится точка, но в первом случае будет напечатан символ "точка", образ которого возьмется из ПЗУ, а во втором случае точка будет нарисована согласно Вашей команде PLOT. Вы, должно быть, не удивитесь, что обе точки спокойно уживаются вместе на одном экране, но на самом деле это могут делать далеко не все компьютеры. В большинстве машин нельзя одновременно воспроизводить на экране и символьную и графическую информацию. Для них существуют совершенно разные режимы

работы. Чтобы напечатать символы должен быть включен "символьный" экран, а чтобы нарисовать что-либо, даже простую точку, должен быть включен графический экран и вместе на одном экране они не уживаются. Конечно, программисты обходят эти проблемы и в графических режимах символы тоже изображаются, но, грубо говоря, они не "печатаются", а "рисуются" специально созданной для этой цели программой (так называемым драйвером экрана).

Итак, Ваш компьютер может одновременно работать и в символьном режиме и в графическом. Это колоссальное удобство, которое мгновенно было оценено производителями программного обеспечения и компьютер, первоначально предназначавшийся для обучения школьников основам программирования, в кратчайшие сроки был оснащен изобилием всевозможнейших программ и превзошел самые смелые ожидания своих разработчиков. Фирмы, выпускающие программное обеспечение конечно были заинтересованы в работе под самый популярный компьютер. Покупатели, в свою очередь, были заинтересованы в приобретении наиболее обеспеченной программами машины, что и делало ее популярной. Так родился кумулятивный эффект и в значительной степени он связан с особенностями экрана компьютера.

Вы, очевидно, знаете, что в один ряд на экране помещаются 32 символа и таких рядов может быть до 22-х. Это означает, что экран имеет разрешение 32 X 22 символа. В то же время, Вы знаете, что экран имеет 256 точек по горизонтали и 172 точки по вертикали, то есть при работе с графикой его разрешающая способность 256 X 172. Такую работу иногда называют работой с графикой высокого разрешения или еще короче "высокой графикой" в отличие от символьной графики.

А теперь, прежде чем двигаться вперед, давайте рассмотрим раскладку экрана компьютера и одновременно договоримся о терминологии, чтобы далее на страницах этой и последующих книг употреблять единые термины.

1.2 РАСКЛАДКА ЭКРАНА

1.2.1. Раскладка алфавитно-цифрового экрана.

Выше мы сказали, что на экране может быть до 22-х символьных рядов. Это не совсем так. На самом деле на полном экране может быть конечно до 24-х символьных рядов, но если Вы работаете на БЕЙСИКе, то подчиняетесь той программе-интерпретатору, которая "зашита" в ПЗУ компьютера, а она использует нижние 2 ряда для собственных сообщений к Вам. Это могут быть сообщения об ошибках или более приятное сообщение "О.К." о том, что все в порядке. Эти нижние два ряда называют системным окном, а верхние 22 строки - Главным экраном.

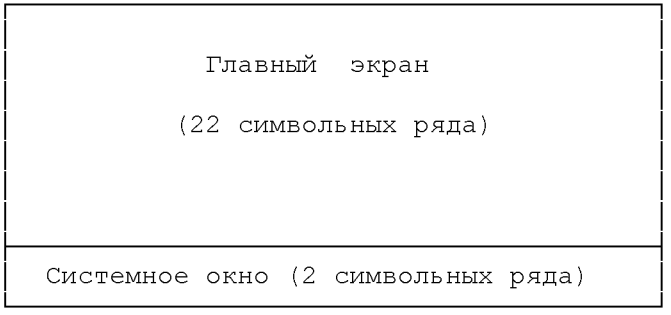


Рис. 1

Вот поэтому-то и получается, что при работе на БЕЙСИКе Вам доступны только 22 символьных ряда или 176 точек по вертикали при работе с графикой.

Если бы Вы работали в машинном коде, то могли бы отключиться от ПЗУ компьютера и тогда могли бы использовать все 24 символьных ряда или 192 точки по вертикали при работе с графикой высокого разрешения, но при этом не смогли бы получать сообщений от компьютера о ходе работы программы и любая ошибка приводила бы либо к "зависанию" программы, либо к ее сбросу.

Возможен еще третий вариант, когда Вы работает в БЕЙСИКе,

но для ускорения тех или иных операций над экраном применяете свои процедуры, написанные в машинном коде, а потом опять возвращаетесь в БЕЙСИК (такой подход мы рассмотрим в Главе 3). В этом случае Вам также лучше оставить нижние 2 ряда в покое и не трогать их.

Есть еще и четвертый подход, когда написанная Вами программа может быть полностью независимой от БЕЙСИКа и исполнена в машинном коде, но для упрощения изображения графических примитивов (точек, линий, дуг, окружностей) Вы используете стандартные процедуры из ПЗУ, вызывая их по известному Вам адресу (приемы будут даны в Главе 2). В этом случае Вам лучше также не вторгаться в системное окно, поскольку эти процедуры могут проверять координаты позиции печати и не работать с тем, что для них не предусмотрено.

Далее мы будем рассматривать экран состоящим как бы из 24-х рядов, помня, что на БЕЙСИКе нам доступны только 22.

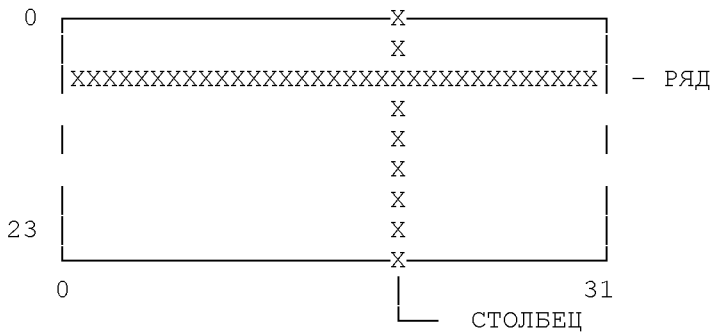


Рис. 2

Итак, экран состоит из 24-х символьных рядов по 32 символа в каждом. Ряды нумеруются сверху вниз от 0 до 23-го. С другой стороны, мы можем считать, что экран состоит из 32-х столбцов, которые нумеруются слева направо от 0 до 31-го. Таким образом, позиция с координатами (0,0) для экрана с низким разрешением находится в левом верхнем углу.

Поскольку в одном ряду можно расположить до 32-х символов, то говорят, что в ряду содержится 32 знакоместа.

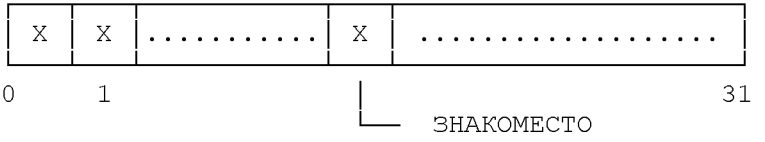


Рис. 3

Рассмотрим теперь, что же представляет из себя каждое знакоместо. Оно состоит из 8 линий по 8 точек в каждой. Эти точки называют пикселами (PICture CELL).

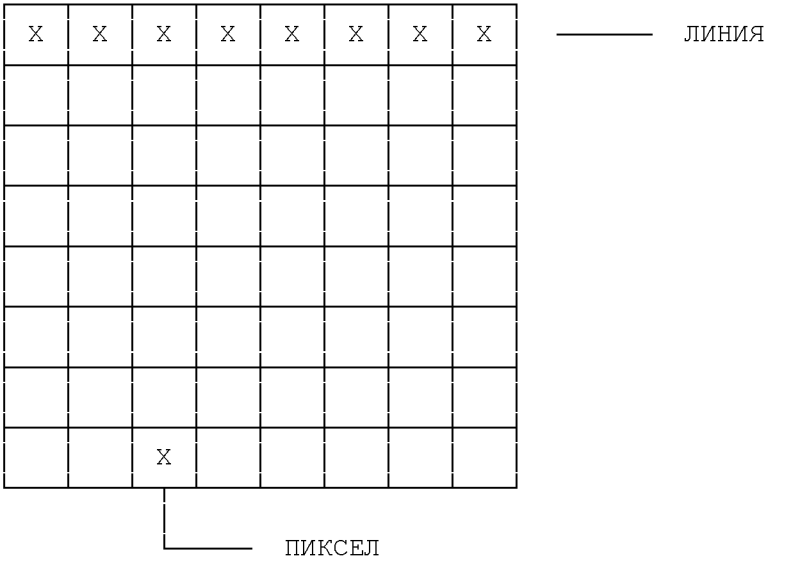


Рис.4

Каждой линии каждого знакоместа на экране Вашего дисплея соответствует один байт в оперативной памяти Вашего компьютера. Таким образом, для того, чтобы запомнить всю информацию, содержащуюся в одном знакоместе, необходимо затратить 8 байтов опе-

ративной памяти. С другой стороны, чтобы заполнить на экране одно знакоместо, должно быть прочитано содержимое восьми байтов памяти. Заметьте, что здесь мы уже не говорим ОПЕРАТИВНОЙ ПАМЯТИ, поскольку читать можно не только из нее, но и из ПЗУ, а именно оттуда и берутся 8 байтов, когда на экране изображается стандартный символ, например буква "А". Эти 8 байтов, из которых строится изображение символа, часто еще называют ШАБЛОНОМ символа.

Если линия - это байт в памяти, то что же такое пиксел? Вы наверное знаете, что байт состоит из восьми битов, каждый из которых может быть включен или выключен (равен единице или нулю). Поэтому каждому пикселу в линии соответствует один бит в байте. Получается, что карта включения битов определяет раскладку включенных пикселов в линии.

Если, например, байт равен 175, то его побитовая раскладка такова:

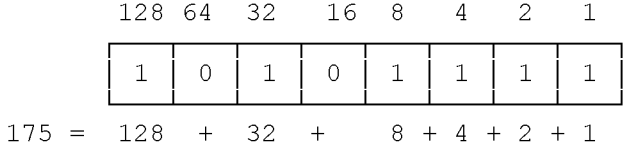


Рис. 5

Совершенно естественно, что линия в знакоместе, которой соответствует байт 175, будет иметь вид:

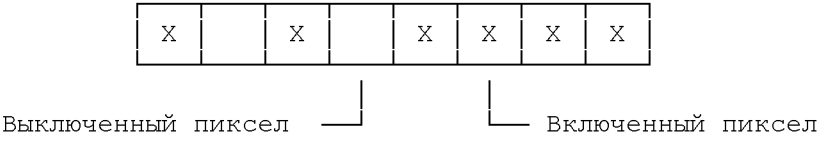


Рис. 6

Сравните Рис.5 и Рис. 6 и убедитесь в их соответствии.

Здесь мы столкнулись с интересной особенностью. Дело в том, что для графических работ, в отличие от математических, неважно, какое значение имеет байт, важна его побитовая раскладка, то есть как бы обрабатываются они не в соответствии со своими значениями, а в соответствии со своей конструкцией.

Обратите внимание на интересное обстоятельство, связанное с особенностью шаблонов стандартных символов. Дело в том, что для того, чтобы текст на экране хорошо читался, между символами должны быть какие-то зазоры. Поэтому в знакоместе 8 X 8 правая колонка пикселов у символов как правило выключена. А это значит, что во всех линиях правый пиксел выключен и в соответствующих им байтах самый младший бит равен нулю (см.рис.5). Если же этот бит равен нулю, то байт будет иметь только четное значение. Поэтому, если при просмотре машинного кода фирменных программ Вы найдете последовательность из нескольких сотен четных байтов, то можно почти уверенно предположить, что здесь расположены шаблоны шрифтового набора. Совсем нетрудно написать БЕЙСИК-программу для поиска, скажем, пятидесяти встретившихся подряд четных байтов и быстро определить, где может находиться шрифт (в этом случае нулевой байт тоже условно отнесем к четным).

Остался еще неисследованным вопрос о том, какой цвет имеет включенный пиксел и какой цвет имеет на экране пиксел, который выключен. Укажем пока только, что включенный пиксел имеет цвет INK, а выключенный - цвет PAPER и вернемся к этому вопросу, когда будем заниматься цветом.

1.2.2. Раскладка экрана в графике высокого разрешения.

Тридцать два знакоместа по ширине экрана дают 256 (32 X 8) пикселов - такова ширина экрана в "высокой" графике.

Двадцать четыре ряда по восемь пикселов в каждом знакоместе дают нам 192 пиксела по вертикали, но поскольку нижние две строки в БЕЙСИКе недоступны, то считают, что предельный размер по вертикали равен 22 X 8 = 176 пикселов.

За начало отсчета принимают левый нижний угол Главного экрана (обратите внимание на слово "Главного", то есть системное окно отбрасывается).

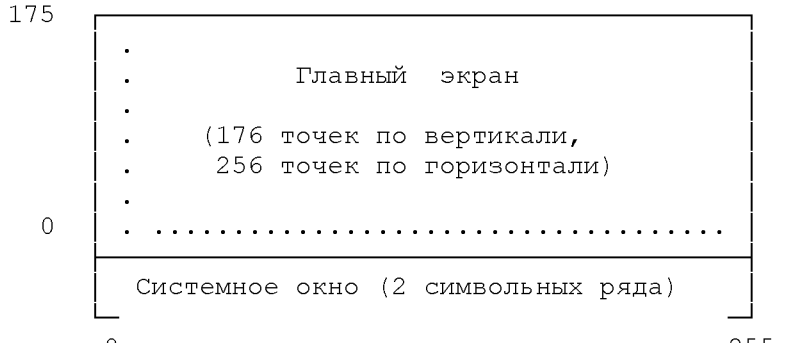


Рис. 7

Таким образом, точка с координатами (0,0), если они заданы в пикселах, определяет левый нижний угол Главного экрана. Тот факт, что вертикальная координата в знакоместах отсчитывается сверху вниз, а в пикселах - наоборот снизу вверх иногда приводит к путанице начинающих программистов, впрочем с набором практики это быстро проходит.

В заключение этого параграфа давайте для справки прикинем сколько байтов оперативной памяти необходимо для хранения пиксельного (монохромного - без цвета) изображения всего экрана. Если на каждое знакоместо нам необходимо восемь байтов, то на экранный ряд - 32 X 8 = 256 байтов, а на весь экран - 24 X 256 = 6144 байта.

1.2.3. Раскладка экрана цветовых атрибутов.

Еще одной особенностью "Синклер"-совместимых компьютеров является необычный подход К.Синклера к вопросам, связанным с цветом. Дело в том, что если каждую точку красить в свой цвет, то оперативной памяти компьютера уже ни на что хорошее не

останется. Поэтому им было принято гениальное решение - цвет сделать с низким разрешением. То есть для каждого знакоместа возможен только один цвет INK (для включенных пикселов) и один цвет PAPER (для выключенных). Иначе говоря, Вы не можете иметь в одном знакоместе более двух цветов одновременно. Что же это дало?

Поскольку всего возможно только 8 цветов, то для задания цвета INK в одном знакоместе достаточно трех битов:

Таблица 1

0	=	000	-	черный	BLACK
1	=	001	-	синий	BLUE
2	=	010	-	красный	RED
3	=	011	-	пурпурный	MAGENTA
4	=	100	-	зеленый	GREEN
5	=	101	-	голубой	CYAN
6	=	110	-	желтый	YELLOW
7	=	111	-	белый	WHITE

Цвета были выбраны отнюдь не случайно, а исходя из тех соображений, что на экране черно-белого телевизора они должны во-первых хорошо различаться, как разные оттенки серого, а во-вторых, их яркость от черного до белого должна постепенно возрастать.

Точно так же для задания 8 цветов PAPER нам тоже достаточно трех битов на одно знакоместо. Если теперь каждому знакоместу отвести по одному байту для хранения параметров цвета (а в байте 8 битов), то остается еще пара свободных битов, один из которых отдан признаку яркости (BRIGHT), а другой - признаку мигания (FLASH). Таким образом, для каждого знакоместа может быть установлен свой цвет INK, свой цвет PAPER и свои режимы BRIGHT и FLASH. Эти четыре параметра называют цветовыми АТРИБУТАМИ.

Использование атрибута BRIGHT эквивалентно расширению

палитры цветов в два раза, то есть компьютер имеет как бы 16 цветов. Оценим теперь расход памяти на цветовые атрибуты:

24 ряда по 32 знакоместа = 768 байтов - совершенно ничтожный расход памяти для такого богатства красок, которое мы нередко видим в хороших игровых программах.

После выхода первых "Спектрумов" не было пределов восторженным отзывам. Добиться имитации многоцветной графики высокого разрешения с таким ничтожным расходом памяти - это великое достижение (за которым конечно скрывается иллюзия, т.к. цвет не имеет высокого разрешения, а только имитирует его за счет высокого разрешения пиксельной графики).

Общий расход памяти на весь экран с цветом составляет: 6144 + 768 = 6912 байтов, - что очень и очень немного по сравнению с машинами конкурентов.

За этим фактом быстро открылась еще одна особенность. Поскольку процессор имеет ограниченную скорость работы, то обслужить (перестроить) экран он может тем быстрее, чем меньшую память имеет экранная область и потому фирмы, выпускающие программное обеспечение, легко смогли освоить динамическую графику и добиться превосходных по тем временам успехов в анимации изображений (так на Западе называют мультипликацию).

Итак, каждый байт атрибутов соответствует одному знакоместу экрана и несет информацию о цветах INK, PAPER и признаках BRIGHT и FLASH. При этом побитовая раскладка этих параметров в байте выглядит так:

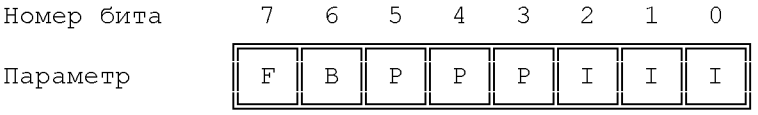


Рис. 8

1.2.4. Основные проблемы при работе с графикой.

Конечно же самая главная проблема при работе с графикой - где взять художественный талант и творческое вдохновение но, здесь мы Вам помочь ничем не сможем - сами их не имеем, так что на первых порах будем считать, что у Вас все это есть, а если и нет, то попробуем обойтись без них, заменив творческое вдохновение на кропотливый поиск методом проб и ошибок. Мы же остановимся на чисто программистских проблемах, связанных с графикой.

1. Быстродействие. Работа с графикой требует быстрой переброски значительных объемов информации (килобайтов) из одних участков оперативной памяти в другие. Здесь Вам поможет применение процедур, записанных в машинном коде или прямое обращение к процедурам, содержащимся в ПЗУ.

2. Вторая проблема - ограниченный объем памяти компьютера. Так как графические изображения требуют больших объемов памяти, приходится решать и этот вопрос. Есть приемы по борьбе с этой проблемой. Во-первых, стараются использовать для графики не весь экран, а только его часть (необходимую и достаточную для достижения поставленной задачи). Во-вторых, стараются по-возможности шире применять вычислительные процессы для создания графических образов - в этом случае графическое изображение не хранится в памяти для выдачи его в нужное время на экран, а строится по заданному алгоритму (векторная графика) именно в тот момент, когда это надо. В-третьих, когда все-таки надо хранить десятки изображений, ранее подготовленных в графическом редакторе, применяют методы компрессии при сохранении рисунка и декомпрессии при выдаче его на экран.

3. Третья проблема является чисто программистской. Она состоит в том, что надо хорошо знать как структура экрана соответствует структуре оперативной памяти. Грубо говоря, надо знать какие адреса в памяти каким координатам на экране соответствуют. А поскольку координаты могут быть заданы в знакоместах или в пикселах, то фактически эту проблему надо решать дважды, а если еще добавить цветовые атрибуты, то и

трижды.

Однако, знать это соответствие еще не достаточно. Надо еще знать методы и приемы, с помощью которых компьютер может рассчитать эти адреса по заданным координатам, надо знать приемы и алгоритмы, с помощью которых данные извлекаются из недр оперативной памяти и распределяются по этим адресам.

К счастью, если Вы работаете в БЕЙСИКе, то забивать себе этим голову не надо. Смело давайте команду PRINT AT, PLOT x,y, DRAW... и т.п. и интерпретатор БЕЙСИКа, "зашитый" в ПЗУ, сам рассчитает, что нужно в какие ячейки памяти заслать, чтобы Вы на экране получили то, что хотите. Совсем иное дело, если Вы будете работать в машинных кодах - тогда читайте вторую и третью главы этой книги, а пока вернемся к БЕЙСИКу.

1.3 КОМАНДЫ БЕЙСИКА, СВЯЗАННЫЕ С ГРАФИКОЙ.

1.3.1 Команды управления цветом и режимами печати.

CLS (CLEAR SCREEN).

Эта команда служит для очистки экрана. Она обнуляет содержимое ячеек той области памяти, в которой хранится экранное изображение (с 16384 по 22527). Таким образом, в результате ее работы с экрана стирается пиксельное изображение. Несколько по-другому она обходится с цветовыми атрибутами. В результате этой команды цвет может не только остаться, но и даже измениться. Дело в том, что информацию о цвете включенных (INK) и выключенных (PAPER) пикселов компьютер черпает из системной переменной ATTR_P (23693 = 5C8DH), в которой установлены "постоянные" значения этих атрибутов. (Бывают еще "временные" значения, о которых мы скажем чуть ниже). Формат, в котором атрибуты INK, PAPER, BRIGHT и FLASH хранятся в системной переменной ATTR_P соответствует Рис 8.

Очень часто команду CLS используют не для того, чтобы очистить экран, а для того, чтобы установить на экране постоянные цветовые атрибуты. Именно для этой цели команда CLS широко применяется в первых строках БЕЙСИК-программы, например:

```
10 INK 6: PAPER 1:
20 CLS
...................
```

INK n

Этот оператор служит для задания цвета, которым на экране изображаются символы или графические элементы - одним словом, он определяет цвет включенных пикселов. После оператора INK должен стоять параметр n, который может быть как целым числом, так и выражением. Этот параметр задает цвет и может принимать значения от 0 до 9. Для INK от 0 до 7 кодовые значения цветов показаны в табл. 1 на стр.17. Но возможны еще INK 8 и INK 9.

Цвет INK 8 называют "транспарантным". В этом случае символ на экране печатается с тем цветом, который существовал в данном знакоместе до этой печати. Очень удобно, когда надо что-то напечатать по готовому цветному изображению. Не надо для каждого знакоместа перезадавать новый цвет.

Цвет INK 9 называют "контрастным". В этом случае символ на экране печатается с таким цветом, который наилучшим образом контрастирует с тем цветом PAPER, который имеется в данном знакоместе в данный момент. Это тоже очень удобно при печати по готовому изображению.

Если посмотреть на рис. 8, то видно, что для задания цвета INK используются три младших бита, с помощью которых можно задать цвета от 0 до 7. Остается вопросом как можно хранить цвета INK 8 и INK 9.

Для хранения постоянных "транспарантных" атрибутов, служит системная переменная MASK_P (23694 = 5C8EH). В ней должны быть включены те биты, которые соответствуют "транспарантным" атрибутам. Так, для INK 8 должны включиться биты 0...2 (три младших бита).

Для того, чтобы компьютер знал о том, что Вы избрали постоянный цвет INK "контрастным", служит другая системная переменная - P_FLAG (23697 = 5C91H). В ней для этого выделен 5-ый бит, который включается при подаче команды INK 9.

Оператор INK может применяться двумя способами. Во-первых, он может быть самостоятельной командой и в этом случае оказывает длительное действие - не только на ближайшую команду печати или рисования, но и на все последующие, пока не будет изменен. По такой команде изменяется содержимое упомянутой выше системной переменной ATTR_P (23693 = 5C8DH), определяющей установку "постоянных" цветовых атрибутов.

Во-вторых, оператор INK может быть не самостоятельной командой, а частью команды PRINT. В этом случае INK называют не

командой, а "квалификатором оператора PRINT". Здесь он задает не "постоянные" значения цветового атрибута, а только "временные", которые действуют только для данного оператора PRINT и все. "Временные" значения хранятся в другой системной переменной - ATTR_T (23695 = 5C8FH), но формат имеют тот же (см. рис. 8). INK n в качестве квалификатора PRINT требует, чтобы после него обязательно стояла точка с запятой.

Сравните:

```
10 INK 7: PRINT "ZX-Spectrum"
```

и

```
10 PRINT INK 7; "ZX-Spectrum"
```

Определять "временные" атрибуты можно не только с параметром n от 0 до 7. Здесь также возможно задание цвета "транспарантным" (INK 8) и "контрастным" (INK 9).

Для хранения временных "транспарантных" атрибутов, служит системная переменная MASK_T (23696 = 5C90H). В ней должны быть включены те биты, которые соответствуют "транспарантным" атрибутам. Так, для INK 8 должны включиться биты 0...2.

Для того, чтобы компьютер знал о том, что Вы избрали временный цвет INK "контрастным", в системной переменной P_FLAG (23697 = 5C91H) выделен 4-ый бит, который включается, когда в операторе PRINT встречен квалификатор INK 9; .

PAPER n

Оператор PAPER определяет цвет выключенных пикселов экрана. За ним также идет параметр n (0...9). Почти все сказанное выше об операторе INK относится и к оператору PAPER. Он тоже может быть самостоятельной командой и задавать "постоянные" значения, действующие до тех пор, пока не будут изменены, а может быть и квалификатором оператора PRINT и задавать временные значения, действующие только на этот оператор PRINT.

```
10 PAPER 2: CLS     -          самостоятельная команда.
20 PRINT INK 6; PAPER 1; "ZX-SPECTRUM" -  квалификатор.
```

Один оператор PRINT может иметь сколько угодно квалификаторов - они должны отделяться друг от друга точкой с запятой.

Естественные отличия PAPER от INK выражаются в том, что информация об их установках хранится в системных переменных в разных битах. Так, для цвета PAPER они таковы:

а) постоянные значения:

- параметр цвета от 0 до 7 - в битах 3...5 системной переменной ATTR_P (23693 = 5C8DH);

- для "транспарантного" цвета PAPER 8 должны быть включены биты 3...5 системной переменной MASK_P (23694 = 5C8EH);

- для цвета PAPER, контрастного к тому цвету INK, который уже имеется в данном знакоместе, т. е. для команды PAPER 9 включается бит 7 флаговой системной переменной P_FLAG (23697 = 5C91H).

б) временные значения:

- параметр цвета от 0 до 7 - в битах 3...5 системной переменной ATTR_T (23695 = 5C8FH);

- для PAPER 8 должны быть включены биты 3...5 системной переменной MASK_T (23696 = 5C90H);

- значению PAPER 9 соответствует включенный бит 6 системной переменной P-FLAG (23697 = 5C91H).

BRIGHT n

Этот оператор задает яркость при печати в знакоместе. Параметр n может принимать значения 0,1 и 8.

BRIGHT 0 - яркость выключена.
BRIGHT 1 - яркость включена.
BRIGHT 8 - атрибут яркости при печати или рисовании в данном знакоместе устанавливается таким же, каким он был до этого.

Он также, как и PAPER и INK может быть самостоятельной командой, задающей постоянное значение атрибуту яркости, а может быть и квалификатором оператора PRINT и задавать временное значение.

а) постоянное значение:

- параметр яркости 0 или 1 - устанавливается в 6-ом бите системной переменной ATTR_P (23693 = 5C8DH);

- для "транспарантного" значения BRIGHT 8 должен быть включен биты 6 системной переменной MASK_P (23694 = 5C8EH);

б) временные значения:

- параметр яркости 0 или 1 - хранится в 6-ом бите системной переменной ATTR_T (23695 = 5C8FH);

- для "транспарантного" значения BRIGHT 8 должен быть включен бит 6 системной переменной MASK_T (23696 = 5C90H);

FLASH n

Этот оператор задает режим мигания пикселов в знакоместе. Включенные пикселы выключаются (принимают цвет PAPER), а выклю-

ченные включаются (принимают цвет INK) и так далее. Параметр n, как и для оператора BRIGHT может принимать значения 0,1 и 8.

FLASH 0 - режим мигания отключен.
FLASH 1 - режим мигания включен.
FLASH 8 - режим мигания при печати или рисовании в данном знакоместе устанавливается таким же, каким он был до этого.

Оператор FLASH n может образовывать самостоятельную команду или квалификатор оператора PRINT:

а) постоянное значение:

- режим мигания 0 или 1 - устанавливается в 7-ом бите системной переменной ATTR_P (23693 = 5C8DH);

- для "транспарантного" значения FLASH 8 должен быть включен бит 7 системной переменной MASK_P (23694 = 5C8EH);

б) временные значения:

- режим мигания 0 или 1 - хранится в 7-ом бите системной переменной ATTR_T (23695 = 5C8FH);

- для "транспарантного" значения FLASH 8 должен быть включен бит 7 системной переменной MASK_T (23696 = 5C90H);

INVERSE n

Оператор INVERSE задает режим печати на экране "с инверсией" символов или без нее. В отличие от INK, PAPER, BRIGHT и FLASH этот оператор не имеет соответствующего цветового атрибута. То есть, в памяти компьютера не запоминается, какому знакоместу какой режим INVERSE соответствует. Одним словом, это

только режим печати и не более того.

INVERSE 0 - режим выключен.
INVERSE 1 - режим включен.

При печати "с инверсией" включенные пикселы изображаются выключенными, т.е. имеют цвет PAPER и наоборот - выключенные изображаются включенными и имеют цвет INK.

Также, как и прочие рассмотренные команды управления цветом, INVERSE может иметь постоянное действие (команда INVERSE n) или временное действие (квалификатор INVERSE n; оператора PRINT).

Информация об установленном режиме INVERSE хранится во флаговой системной переменной P_FLAG (23697 = 5C91H). Бит 2 этой переменной определяет временную установку режима (0 или 1), а бит 3 - постоянную установку.

OVER n

Оператор OVER задает режим печати на экране "с наложением" или без него.

OVER 0 - режим выключен.
OVER 1 - режим включен.

Таблица 2

	OR				XOR			
Бит 1	0	1	0	1	0	1	0	1
Бит 2	0	0	1	1	0	0	1	1
Результат	0	1	1	1	0	1	1	0

Наложение осуществляется по логическому принципу "ИСКЛЮЧАЮЩЕГО ИЛИ" - обозначется XOR. Если оба соответствующих пиксела были выключены, то результирующий пиксел - выключен. если один из составляющих пикселов был включен, то результирующий тоже включен, но если оба пиксела были включены, то результирующий выключен. В этом заключается отличие действия логической команды XOR от команды OR. Сравните их действие по таблице 2.

Таким образом, если Вы в режиме OVER 1 напечатаете некий символ поверх самого же себя, то получите пробел. В этом, кстати, заключается довольно эффективный прием стирания. Можно, конечно, стереть изображение в знакоместе напечатав пробел, но это не всегда возможно. Например, Вам надо перемещать на экране Вашего героя по какому-то фону. Перемещение состоит в последовательности следующих действий:

- печать в текущей координате;
- стирание в текущей координате;
- печать в соседней координате;
- и так далее.

Первую печать Вашего героя поверх фона Вы сделаете без проблем, включив режим OVER 1, а вот стирание с помощью пробела здесь не пройдет, т.к. "погибнет" и нижележащее (фоновое) изображение. На помощь приходит повторная печать того же изображения в том же месте в режиме OVER 1. Теперь наложенное изображение исчезнет, а фоновое (исходное) останется без изменений.

Как и INVERSE, этот оператор не имеет соответствующего цветового атрибута. То есть, в памяти компьютера не запоминается, какому знакоместу какой режим OVER соответствует. Это тоже только режим печати. Как и INVERSE, оператор OVER n может быть самостоятельной командой, а может использоваться как квалификатор оператора PRINT.

Информация об установленном режиме OVER хранится тоже во флаговой системной переменной P-FLAG (23697 = 5C91H). В таблице

3 приведена побитовая карта этой системной переменной:

Раскладка системной переменной P-FLAG.
Таблица 3.

Бит	Назначение	Характер
0	Указатель режима OVER	Временный
1	Указатель режима OVER	Постоянный
2	Указатель режима INVERSE	Временный
3	Указатель режима INVERSE	Постоянный
4	Указатель режима INK 9	Временный
5	Указатель режима INK 9	Постоянный
6	Указатель режима PAPER 9	Временный
7	Указатель режима PAPER 9	Постоянный

BORDER n

Команда BORDER n определяет цвет, в который окрашивается внешняя рамка (бордюр) экрана. Этих цветов может быть 8. У этой команды есть две характерных особенности, о которых необходимо помнить.

Во-первых, как ни странно, к бордюру относятся и нижние 2 строки экрана ("системное окно", о котором мы говорили выше в разделе 1.2.1), поэтому плохо рассчитанная команда BORDER может подпортить то, как выглядят эти 2 строки.

Во-вторых, окрашивание бордюра экрана производится не так, как окрашивание самого экрана. Если для экрана параметры цвета по каждому знакоместу хранятся в соответствующих ячейках памяти и процедуры ПЗУ, обслуживающие экран, черпают информацию оттуда, то бордюр исполняется не процедурами ПЗУ, а аппаратными средствами. С точки зрения архитектуры компьютера экран является одним внешним устройством, а бордюр - как бы совсем другим, которое подключено через внешний порт 254 (FE). Поподробнее мы расскажем об этом в главе 3.

Поэтому окрасить бордюр из БЕЙСИКа можно не только командой BORDER, но и выдав байт на внешний порт с помощью команды OUT, например:

```
OUT 254,1
```

С другой сторны, между командами BORDER n и OUT 254,n есть и существенная разница. Цвет бордюра может иметь как постоянный, так и кратковременный характер. Команда BORDER задает цвет "постоянным". Он запоминается в системной переменной BORDCR (23624 = 5C48H). Команда же OUT 254,n имеет очень кратковременный характер. Фактически она срабатывает только по фронту сигнала, пошедшего на внешний порт, после чего цвет опять восстановится в соответствии с BORDCR. Так что изменить цвет бордюра командой OUT, не изменив системную переменную BORDCR не удастся. О том, как используется способность бордюра менять свой цвет по фронту сигнала, Вы можете судить по тем цветным полосам, которые Вы видите на экране в момент загрузки-выгрузки программ. Разбору управления цветными полосами на бордюре мы посвятили раздел в нашей более ранней книге "Практикум по программированию в машинных кодах".

TAB n

Оператор TAB n (n=0...31) в качестве самостоятельной команды не применяется. Он служит в качестве квалификатора оператора PRINT и определяет, с какой позиции в текущей строке надо выполнить печать.

Попробуйте дать прямую команду:

```
PRINT TAB 10; "ZX-Spectrum".
```

Можете заменить параметр 10 например на 20.

AT y,x

Как и квалификатор TAB, AT y,x самостоятельно без оператора PRINT не применяется, но является более мощным и исполь-

зуется значительно чаще. Он задает координату позиции печати (знакоместа) на экране. Параметр y задает координату по вертикали (номер экранного ряда, в котором надо выполнить печать) от 0 до 21. Параметр x задает координату по горизонтали (номер экран- ного столбца) от 0 до 31.

Пример: PRINT INK 6; PAPER 1; AT 10,12; "ZX-Spectrum" .

В качестве параметров y и x могут стоять не обязательно числа, но и арифметические, логические или алгебраические выражения, если они дают целочисленный результат, не выходящий за допустимые пределы по x и y.

Иногда в программах бывает нужно установить позицию печати в какую-то точку, но саму печать при этом пока делать не требуется. Это можно сделать с помощью оператора AT, дав команду напечатать пустую строку, например:

PRINT AT (10,12); ""

Прочие квалификаторы оператора PRINT.

Рассматривая команды и квалификаторы, управляющие цветом и координатами того, что воспроизводится на экране, мы должны также сказать еще о трех квалификаторах. Это "," (запятая), ";" (точка с запятой) и "'" (апостроф).

В отличие от прочих ранее рассмотренных квалификаторов типа INK, PAPER, AT... и др. эти квалификаторы не надо отделять друг от друга, поскольку они сами являются разделителями. Рассмотрим их по-порядку.

"Точка с запятой".

Когда в операторе PRINT встречается этот символ-разделитель, это означает указание компьютеру выполнить печать в ближайшем справа знакоместе от предыдущей позиции печати, то

есть печатать в сплошную строку. Только если текущая строка закончилась, компьютер перейдет к печати на следующей строке. Этот квалификатор встречается наиболее часто. Попробуйте:

```
PRINT AT 10,14; "1"; "2"; "3"; "4"
```

"Запятая"
~~~~~~~~~

Мысленно разделите экран по вертикали на две равные части. Представьте себе, что каждый ряд на экране состоит из двух половин - левой и правой (по шестнадцать знакомест в каждой).

Если в операторе PRINT компьютер встретит этот квалификатор, то очередную печать он начнет от начала правой половины экранного ряда, если предшествующая печать была на левой половине. Если же печать уже была на правой половине, то он перейдет на новую строку и начнет печать с ее начала. Таким образом, действие этого квалификатора эквивалентно TAB 16. Попробуйте:

```
PRINT AT 10,14; "1", "2", "3", "4"
```

"Апостроф"
~~~~~~~~~~

Это самый простой из всех квалификаторов - он дает команду начать печать с новой строки.

```
PRINT AT 10,14; "1" ' "2" ' "3" ' "4"
```

1.3.2 Команды печати графических примитивов.

Из стандартного БЕЙСИКа компьютера "ZX-Spectrum" Вам доступны четыре типа графических примитивов: точка, линия, дуга, окружность.

```
PLOT x,y
~~~~~~~~~
```

Команда служит для печати точки на экране в координатах y (по вертикали) и x (по горизонтали). $0 < x < 255$; $0 < y < 175$.

Мы уже говорили о том, что когда координаты заданы не в знакоместах, а в пикселах, то отсчет по вертикали идет снизу вверх, причем нижние 2 ряда, являющиеся "системным окном" из системы координат выпадают.

С оператором PLOT могут быть использованы и квалификаторы оператора PRINT, такие как INK, PAPER, OVER и т.п. Интересно отметить, что печать точки по точке при включенном режиме OVER 1 приводит к ее стиранию. Например:

```
10 PLOT 120, 120
15 PAUSE 10
20 PLOT OVER 1; 120,120
30 PAUSE 10
40 GO TO 10
```

Команда PLOT имеет одно побочное действие. После ее исполнения в системную переменную COORDS (23677=5C7DH) заносятся координаты последней напечатанной точки.

DRAW x,y или DRAW x,y,z

Команда служит для рисования на экране отрезков прямых или дуг окружностей и имеет два формата. DRAW x, y - служит для рисования отрезков прямых. DRAW x,y,z - для рисования дуг окружностей. И та и другая команды проводят линию от той точки, которая была напечатана последней и координаты которой хранятся в системной переменной COORDS (23677=5C7DH) к новой точке, которая отстоит от исходной на x пикселов по горизонтали и на y пикселов по вертикали. Таким образом, параметры x и y определяют не координаты конца отрезка (дуги) на экране, а "смещение" конца отрезка (дуги) от его начала и могут быть отрицательными.

Параметр z в команде для рисования дуг окружностей задает фактор кривизны дуги, а точнее говоря угол дуги, измеренный в радианах. Чем больше этот угол, тем больше кривизна дуги. Если он равен нулю, то фактически дуга имеет нулевую кривизну и превращается в прямую линию.

Не очень удобно задавать угол дуги в радианах. В градусах работать все-таки привычнее, поэтому обычно пользуются тем, что "Спектрум" понимает, что такое число "пи" и вместо параметра z подставляют выражение a*PI/180. Получается:

```
DRAW x,y,a*PI/180
```

Здесь параметр "a" можно задавать в градусах.

Обратите также внимание на ряд следующих обстоятельств:

1. Параметры x и y могут быть выражены как числом, так и числовыми или алгебраическими выражениями. Важно только, чтобы после расчета этих выражений результат получался бы целочисленным и чтобы при рисовании отрезка (дуги) не происходил бы выход за пределы экранной плоскости.

2. Для работы команды DRAW совершенно неважно в результате какого действия была получена последняя ранее напечатанная точка - то ли в результате команды PLOT, то ли в результате другой команды DRAW, то ли в результате команды CIRCLE, которую рассмотрим чуть ниже. Для работы DRAW важно только то, что содержится в системной переменной COORDS. Можете даже и сами установить в ней то, что Вам нужно, с помощью команды POKE.

3. Команда DRAW, рисуя линию (дугу), не включает в нее исходную точку, взятую из COORDS. Предполагается, что она уже и так нарисована. При рисовании это действительно не очень важно, но при стирании на это нужно обратить внимание.

4. Закончив работу, команда DRAW оставляет в системной переменной COORDS координаты последней точки. Новая команда DRAW будет работать от той точки, в которой закончила работать предыдущая.

5. Команда DRAW, как и команда PLOT, допускает использование внутри себя квалификаторов оператора PRINT - INK, PAPER, FLASH, BRIGHT, OVER, INVERSE.

CIRCLE x,y,r
~~~~~~~~~~~~

По этой команде на экране изображается окружность с координатами центра x,y, имеющая радиус r. Если радиус равен нулю, то окружность вырождается в обыкновенную точку. Все замечания, сделанные к команде DRAW, могут быть отнесены и к этой команде.

1.3.3 Команды для сканирования экрана.

При работе с графикой иногда необходимо не только что-то напечатать, но и из программы определить что же напечатано в том или ином знакоместе или в той или иной координате. Фактически это задача о сканировании экрана. В БЕЙСИКе Вы имеете ряд возможностей, с помощью которых можно кое-что сделать, хотя надо заметить, что они достаточно редко применяются на практике. Обычно, когда речь идет о необходимости сканирования экрана, мы имеем дело с развитой системой или с достаточно серьезно подготовленной программой. В этих случаях редко используют БЕЙСИК, а применяют вместо него машинный код. Тем не менее, такие возможности в БЕЙСИКе есть и наш обзор был бы неполным, если бы мы на них не указали.

Итак, для сканирования экрана из БЕЙСИКа в Вашем распоряжении есть три функции - POINT, ATTR и SCREEN$. Все они требуют при вызове указания координат и по окончании работы выдают результат, который может быть проанализирован.

POINT x,y
~~~~~~~~~~

Эта очень простая функция проверяет включен или нет пиксел с координатами x,y (0<x<255; 0<y<175). Если он включен - функция возвращает 1, а если выключен - 0.

Простейший способ применения в аркадных играх - для проверки попал ли выпущенный Вами снаряд в цель или нет. После каждого шага перемещения снаряда по экранному полю Вы можете с помощью функции PLOT проверить пиксел, находящийся непосредственно перед ним. Если окажется, что пиксел включен, то значит

там кто-то или что-то есть. Можно выполнять переход GO SUB на подпрограмму, которая разберется кто же там был, начислит очки и изобразит картину взрыва.

Возможны и други пути использования функции POINT. Ниже приведены две несложные программы, которы используют эту функцию для оригинального изображения коротких сообщений.

По запросу впечатайте слово длиной до 6 символов.

```
 10  LET x=0: LET y=0: LET e=2: LET f=160
 20  INPUT "WORD? "; A$
 30  IF LEN A$ >6 THEN GO TO 20
 40  PRINT AT 0,0; A$
 50  FOR s=0 TO LEN A$*8
 60  FOR t=175 TO 168 STEP -1
 70  IF POINT (s,t)=1 THEN GO SUB 200
 80  LET y=y+e
 90  NEXT t
100  LET y=0: LET x=x+e:
110  NEXT s
120  LET f=90: LET e=2: LET x=0: GO TO 50
200  PLOT x,f -(y+e): DRAW 0,e
210  PLOT x,f-y: DRAW e,0
220  PLOT x+e,f-y: DRAW 0,e
230  PLOT x+e,f-(y+e): DRAW -e,0
240  LET e=e+0.5
250  RETURN
```

По запросу впечатайте слово длиной до 10 символов.

```
 10  LET y=0
 20  BORDER 0
 30  PAPER 0
 40  INK 7
 50  CLS
 60  PRINT "WORD?"
 70  INPUT A$
```

```
 80  CLS
 90  IF LEN A$ > 10 THEN GO TO 60
100  PRINT INK 0; AT 0,0; A$
110  PLOT 0,75
120  DRAW 255,0
130  LET  d=LEN A$*8
140  LET x= 127-(d*3/2)
150  FOR s=0 TO d*8
160  FOR t= 168 TO 175
170  IF POINT (s,t)=1 THEN GO SUB 300
180  LET y=y+5
190  NEXT t
200  LET y=0
210  LET x=x+3
220  NEXT s
230  STOP
300  FOR m=1 TO 1 STEP -1
310  CIRCLE x,75+y,m
320  CIRCLE x, 75-y,m
330  NEXT m
340  RETURN
```

ATTR (Y,X)
~~~~~~~~~~

Эта функция позволит Вам определить, какие цветовые атрибуты установлены в интересующем Вас знакоместе с координатами Y,X (0<Y<21; 0<X<31). Как видите, координаты задаются не в пикселах, а в знакоместах.

Функция возвращает число от 0 до 255, из которого можно определить атрибуты INK, PAPER, BRIGHT и FLASH, согласно побитовой раскладке цветовых атрибутов (см. рис. 8). Проще говоря, это число образовалось как:

```
A = INK + PAPER*8 + BRIGHT*64 + FLASH * 128
```

и, соответственно, из него можно вновь рассчитать каждый из этих атрибутов.
~~~~~~~~~~

```
FLASH = INT (A/128):  A = A - FLASH*128:
BRIGHT = INT (A/64):  A = A - BRIGHT*64:
PAPER = INT (A/8):    A = A - PAPER*8:
INK = A
```

Если вернуться к нашему гипотетическому примеру со снарядом, то эта функция может позволить Вам определить во что же попал снаряд. А делают это в простейших играх так:

1. Выбирают черный цвет PAPER в качестве фона игрового экрана.

2. Для себя решают, что все дружеские войска будут изображены желтым цветом, вражеские войска - красным, посторонние предметы могут быть зелеными (растительность) или белыми (железобетонные сооружения).

3. После того, как функция PLOT определила, что перед снарядом что-то есть, функцией ATTR проверяют, какие там включены атрибуты. Определяют цвет INK и решают что делать с этим объектом.

- если цвет желтый - начисляется штраф;
- если он красный - начисляются очки;
- если он зеленый, объект стирается с экрана, в знакоместе печатается пробел;
- если он белый, то ничего не происходит.

Конечно, схема здесь упрощена до предела, но сама концепция проверки ситуации, в которой находится герой программы, с помощью анализа цветовых атрибутов соседних знакомест встречается очень и очень часто.

SCREEN$ (Y,X)

Эта функция задумывалась как еще более мощная, чем ATTR(Y,X). Она выдает символ, который содержится в знакоместе, заданном координатами Y и X (0<Y<21; 0<X<31). Это именно символ, а не число и его можно распечатать например командой

```
PRINT AT 10,10;"X"; AT 0,0; SCREEN$ (10,10)
```

Функция работает следующим образом. Она читает линию за линией все 8 байтов шаблона того изображения, которое находится в заданном знакоместе, а потом сравнивает эти 8 байтов с восьмерками байтов в знакогенераторе. Если там находится равная восьмерка, то можно считать, что символ найден, а если нет, то функция возвращает пустую строку.

К сожалению, она не может распознать символы графики пользователя, т.к. их шаблоны находятся не в знакогенераторе, а в специально для этого отведенной области памяти. Тем более она не в состоянии распознать символы блочной графики, т.к. они вообще не имеют шаблонов в памяти, а всякий раз по мере необходимости конструируются заново по своему номеру. Но делу можно помочь. В разделе 2.10 мы привели универсальную сканирующую программу, которая не только распознает символы графики пользователя, но и символы блочной графики.

Если вернуться к нашему предыдущему примеру со снарядом, она не только определит класс, к которому относится объект, но и точно укажет что это такое, проанализировав его изображение. Так, например, она сможет отличить вражеский танк от самолета и по-разному изобразить сцену поражения.

1.4 ГРАФИКА ПОЛЬЗОВАТЕЛЯ

1.4.1 Работа с графикой пользователя.

Пользователь компьютера, совместимого с ZX-Spectrum может несколько раширить доступный ему набор символов, использовав для этого так называемые символы графики пользователя.

В верхних областях памяти компьютера, начиная с адреса 65368 и по адрес 65535 находится оласть UDG-графики (графики пользователя). Она занимает 168 байтов и, поскольку на каждый символ расходуется по 8 байтов, то есть возможность задать 21 символ.

На адрес начала графики пользователя указывает системная переменная UDG, находящаяся по адресу 23675 (5C7BH). Конец области графики пользователя - это физический конец оперативной памяти компьютера, на который указывает системная переменная P_RAMT (23732 = 5CB4H).

На клавиатуре компьютера эти символы "привязаны" к клавишам от A до U в алфавитном порядке и выдается на экран каждый символ нажатием на соответствующую клавишу, но делать это надо, когда компьютер находится в графическом режиме, т.е. стоит графический курсор "G". Этот режим включается и выключается одновременным нажатием на клавиши CAPS SHIFT и "9".

Вы можете в любое время проверить это например на клавише "A", но ничего интересного, кроме символа "A" не получите, поскольку пока не сформировали в области UDG ни одного шаблона.

Рассмотрим работу с графикой пользователя на конкретном примере. Предположим, Вы хотите изобразить вертолет, имеющий размер 24 X 16 пикселов и занимающий на экране 6 знакомест. Следовательно, наша задача так задать шаблоны шести символов UDG (A,B,C,D,E,F), чтобы распечатанные на экране в следующем порядке, они образовали бы вертолет (рис. 9).

A	B	C
D	E	F

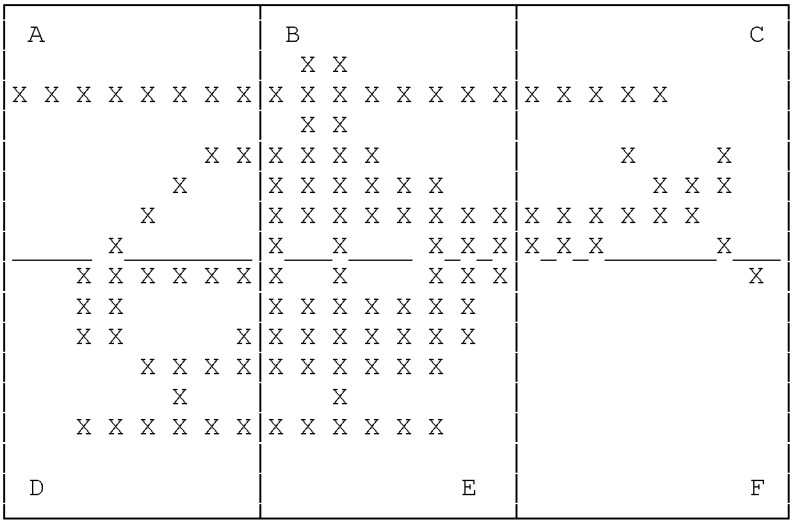


Рис. 9

Рассмотрим подробно как создается шаблон на примере графического символа "B" (рис. 10).

Таким образом, нам достаточно заслать в адреса, начиная с которых располагается символ графики пользователя "B" восемь байтов 0, 96, 255, 96, 240, 248, 255, 167 и клавиша "B" в графическом режиме будет определена. Осталось только разыскать адрес, начиная с которого должен быть размещен этот восьмибайтный шаблон.

Здесь нам поможет "Спектрумовская" функция USR. Она для того и служит, чтобы отыскивать адреса шаблонов символов графики пользователя.

128	64	32	16	8	4	2	1	
								0000 0000 = 0
	X	X						0110 0000 = 96
X	X	X	X	X	X	X	X	1111 1111 = 255
	X	X						0110 0000 = 96
X	X	X	X					1111 0000 = 240
X	X	X	X	X				1111 1000 = 248
X	X	X	X	X	X	X	X	1111 1111 = 255
X		X			X	X	X	1010 0111 = 167

Рис. 10

Команда PRINT USR "b" - даст Вам этот адрес, хотя он Вам и не очень нужен. От Вас ведь требуется только заслать туда 8 байтов. Организуйте цикл:

```
10  FOR i = 0 TO 7
20  READ a: POKE USR "b"+i, a
30  NEXT i
40  DATA 0, 96, 255, 96, 240, 248, 255, 167
```

Точно так же можно задать и остальные символы A,C,D,E,F. Для справки, если Вы захотите изобразить вертолет, приводим их шаблоны.

```
A:  0, 0, 255, 0, 3, 4, 8, 16
C:  0, 0, 248, 0, 18, 14, 252, 226
D:  63, 48, 49, 15, 4, 63, 0, 0
E:  167, 254, 254, 252, 32, 252, 0, 0
F:  1, 0, 0, 0, 0, 0, 0, 0
```

Сам же вертолет будет напечатан например так:

```
100 PRINT AT 10,15 "ABC"
110 PRINT AT 11,15 "CDE"
```

Не забудьте только, что символы от A до E набираются в графическом режиме.

1.4.2. Создание банков символов графики пользователя.

Символы графики пользователя в умелых руках могут дать очень многое. С их помощью можно в кратчайшие сроки сделать модель будущей аркадной игры, разработать полноценную программу для деловых приложений и многое многое другое. Существенный недостаток их состоит в том, что 21 символ - это не очень много, но делу можно помочь. Вы можете, например, иметь несколько заготовленных заранее банков символов и по мере необходимости переключаться с одного банка на другой, что выполняется путем подстановки заданного банка на место текущего.

Каждый банк имеет размер 21*8 =168 байтов и перброска их из того места, где они хранятся, в область, отведенную для символов UDG, может занять определенное время, если эту операцию делать на БЕЙСИКе, поэтому здесь мы используем небольшую процедуру в машинном коде.

Символы UDG хранятся в четырех банках. Программа позволяет записывать содержимое области UDG в эти банки или, наоборот, выполнять загрузку того, что в них находится в область памяти, отведенную для символов графики пользователя, причем делать это можно как из БЕЙСИК-программы, так и с помощью прямой команды. Команды имеют следующий формат:

- для записи данных в банк n (n=1...4):

```
RANDOMIZE USR address: REM Wn
```

- для считывания данных из банка n:

RANDOMIZE USR address: REM Ln

Параметр address - адрес, с которого начинается приведенная ниже программа. С этим адресом связаны и адреса, в которых располагаются четыре банка символов UDG. Вы можете сами задать тот адрес, который Вам удобен, внеся изменения в нижеприведенный код, а можете воспользоваться тем, который уже задан. Нами установлен начальный адрес 64000. Если Вы захотите его изменить, Вам надо будет поменять те значения в строках DATA, которые помечены знаком "*".

Конечно, при наборе строк DATA знак "*" набирать не нужно.

Бейсик-загрузчик.

```
 5   LET addr = 640000
10   CLEAR addr-1
20   FOR k = addr TO addr+16: READ a: POKE k,a: NEXT k
30   FOR k = addr+30 TO addr+47: READ a: POKE k,a: NEXT k
40   FOR k = addr+60 TO addr+77: READ a: POKE k,a: NEXT k

50   DATA    42,  93,  92,  35,  35
60   DATA   126, 254,  87, 202, *30
70   DATA  *250, 254,  76, 202, *60
50   DATA  *250, 201,  35, 126, 214
80   DATA    48,  33,  88, 255,  17
90   DATA     0,*250, 130,  87,   1
100  DATA   168,   0, 237, 176, 201
110  DATA    35, 126, 214,  48,  17
120  DATA    88, 255,  33,   0,*250
130  DATA   132, 103,   1, 168,   0
140  DATA   237, 176, 201
```

Дисассемблер программы:

```
64000   2A5D5C        LD HL,(5C5DH)  ; 5C5DH=23645 DEC. Это ад-
                                     ;рес системной переменной
```

```
                                        ;CH_ADD. Во время работы
                                        ;БЕЙСИК-программы она со-
                                        ;держит адрес очередного
                                        ;символа обрабатываемой
                                        ;БЕЙСИК-строки.
                                        ;Поскольку у нас исполняе-
                                        ;ется RANDOMIZE USR, то
                                        ;эта системная переменная
                                        ;указывает на символ ":"
                                        ;(двоеточие).
 64003       23         INC HL         ;Теперь HL указывает на
                                        ;ключевое слово REM.
 64004       23         INC HL         ;HL указывает на символ,
                                        ;стоящий за REM. Это либо
                                        ;"W", либо "L".
 64005       7E         LD A,(HL)      ;Приняли его в аккумулятор
 64006     FE57         CP 57H         ;Сравнили его с 57H.
                                        ;57H=87 DEC - это код бук-
                                        ;вы "W".
*64008   CA1EFA         JP Z,FA1EH     ;Если это он, то переход
                                        ;на адрес FA1EH = 64030.
 64011     FE4C         CP 4CH         ;Сравнили его с 4CH.
                                        ;4CH=76 DEC - это код бук-
                                        ;вы "L".
*64013   CA3CFA         JP Z,FA3CH     ;Если это он, то переход
                                        ;на адрес FA3CH = 64060.
 64016       C9         RET            ;Возврат в БЕЙСИК.
```

В эту подпрограмму мы попадаем, если была подана команда RANDOMIZE USR addr: REM Wn, то есть речь идет о записи символьного набора из области UDG в n-ый ,банк памяти.

```
 64030       23         INC HL         ;Теперь HL указывает на
                                        ;символ, стоящий после "W"
                                        ;Там должна быть цифра от
                                        ;1 до 4.
 64031       7E         LD A,(HL)      ;Приняли символ этой цифры
                                        ;в аккумулятор.
```

```
 64032     D630        SUB 30H          ;Отняли 30H (48 DEC). Дело
                                        ;в том, что символы цифр
                                        ;начинаются с 48-го кода.
                                        ;Код 48 соответствует
                                        ;цифре "0". Сделав это
                                        ;вычитание, мы получим
                                        ;вместо кода цифры - саму
                                        ;цифру.
 64034   2158FF        LD HL,FF58H      ;Установили в HL указание
                                        ;на адрес FF58H=65368 DEC.
                                        ;В "Спектруме" стандартно
                                        ;с этого адреса начинаются
                                        ;шаблоны символов UDG.
*64037   1100FA        LD DE, FA00H     ;Установили в DE адрес
                                        ;64000.
 64040       82        ADD A,D          ;
 64041       57        LD D,A           ;Теперь DE содержит адрес,
                                        ;равный 64000+256*n, где
                                        ;n -номер банка (1...4).
 64042   01A800        LD BC,00A8       ;Счетчик перемещаемых
                                        ;байтов - A8H=168 DEC.
 64045     EDB0        LDIR             ;Блочное перемещение
                                        ;168 байтов из адреса,
                                        ;на который указывает HL
                                        ;в адрес, на который
                                        ;указывает DE.
 64047       C9        RET              ;Возврат.
```

В эту подпрограмму мы попадаем, если была подана команда RANDOMIZE USR addr: REM Ln, то есть речь идет о записи символьного набора из n-го банка памяти в область UDG.

```
 64060       23        INC HL           ;Теперь HL указывает на
                                        ;символ, стоящий после "L"
                                        ;Там должна быть цифра от
                                        ;1 до 4.
 64061       7E        LD A,(HL)        ;Приняли код этой цифры.
```

```
 64062     D630        SUB 30H         ;Отняли 30H (48 DEC),
                                       ;и получили вместо кода
                                       ;цифры - саму цифру.
 64064   1158FF        LD DE,FF58H     ;Установили в DE указание
                                       ;на адрес FF58H = 65368
                                       ;(это начало области UDG).
*64067   2100FA        LD HL,FA00H     ;Установили в HL указание
                                       ;на адрес 64000.
 64070       84        ADD A,H         ;
 64071       67        LD H,A          ;Теперь в HL содержится
                                       ;адрес 64000+256*n, где
                                       ;n-номер банка (1...4).
 64072   01A800        LD BC,00A8      ;Счетчик перемещаемых
                                       ;байтов - A8H=168 DEC.
 64075     EDB0        LDIR            ;Блочное перемещение
                                       ;168 байтов из адреса,
                                       ;на который указывает HL
                                       ;в адрес, на который
                                       ;указывает DE.
 64077       C9        RET             ;Возврат.
```

Если Вы избрали для хранения этой процедуры адрес ADDR, то первый банк будет храниться, начиная с ADDR + 256, второй - начиная с ADDR + 512 и т.д. Если же Вы желаете изменить исходный адрес ADDR, то Вам надо будет внести изменения в строки, помеченные знаком "*".

2. ЭЛЕМЕНТАРНАЯ ГРАФИКА В МАШИННЫХ КОДАХ

Когда мы слышим сочетание слов "компьютерная графика", то нам сразу представляются сложные многоцветные трехмерные изображения и желательно, чтобы при этом что-то двигалось, лучше, если побольше, побыстрее и как можно более плавно.

Все это, конечно же так, но начинается компьютерная графика, тем не менее, не с этого. Когда Вы даете команду компьютеру PRINT "*" и он это делает, Вы уже работаете с графикой, хотя об этом и не задумывались. Можно считать так, что как только Вы делаете что-то, что приводит к изменению изображения на экране Вашего телевизора или монитора, Вы уже занимаетесь компьютерной графикой, особенно если Вам понятно, почему эти изменения происходят именно так, а не иначе и в какой-то степени можете ими управлять.

Итак, если Вы из БЕЙСИКа напечатаете звездочку на Вашем экране, то Вам потребуется изрядная доля воображения для того, чтобы считать, что это компьютерная графика и убедить своих друзей, что у Вас есть дизайнерские способности. А что, если Вы сделаете то же самое из машинного кода? А если при этом Вы ее не напечатаете, а нарисуете по точкам? Все дело принимает совсем другой оборот, не правда ли? Итак, все дело не в терминах, а в целенаправленности Ваших усилий, в способности задумать что-то и найти способы, как это реализовать.

Если Вы не нашли до сих пор достаточно времени, чтобы освоить программирование в машинных кодах, то Вы не только сузили круг своих технических возможностей, но и ограничили возможности для самовыражения, для дальнейших творческих поисков. Наш пример с печатанием звездочки здесь как раз и служит для того, чтобы дать представление о том, что и в графике значение имеет не только конечный результат, но и путь, который к нему привел.

В этой главе мы попробуем дать Вам те основы, которые необходимы для того, чтобы начать эксперименты с графикой из машинного кода. Как и в любом другом вопросе, связанном с про-

граммированием на "Спектруме", мы не надеемся дать полную и исчерпывающую картину. Как и всегда, "ИНФОРКОМ" видит главную задачу в том, чтобы помочь сделать первый шаг, а дальше Вы сами раскроете свои таланты.

2.1 ПОНЯТИЕ О КАНАЛАХ И ПОТОКАХ

Первое, что нам потребуется, это разобраться с концепцией потоков и каналов "Спектрума". Для многих начинающих пользователей "Спектрума" такие понятия, как каналы и потоки могут звучать, как непонятные жаргонные обозначения, но на самом деле за ними скрывается интересная концепция, которая позволит Вам взять от компьютера то, что другими способами взять не так просто.

Работая в БЕЙСИКе, Вы можете и не задумываться о потоках и каналах, а вот программируя в машинных кодах, без них не обойтись.

Можете представить себе, что канал - это некоторое техническое устройство, используемое для ввода/вывода информации. Надо, правда, оговориться, что канал - не всегда техническое устройство. Каналом, например, может быть файл на диске или, скажем, в памяти Вашего компьютера. В файл ведь тоже можно заносить информацию и можно ее оттуда считывать.

Проще всего представить концепцию каналов и потоков на примере морского побережья с многочисленными заливами и бухтами. Со стороны суши в них впадают многочисленные ручьи и реки. Так вот, эти заливы и бухты - это каналы, а те ручьи и реки, которые в них впадают - это потоки, подключенные к каналам.

Те, кто более глубоко заинтересуются этой концепцией, могут найти информацию в нашем издании "ZX-РЕВЮ" (N12, 1991г., с.227; N 5,6, 1992 г.,с. 111), мы же здесь рассмотрим этот вопрос в минимально необходимом объеме.

Стандартными каналами "Спектрума" для вывода информации являются каналы "K" - нижние две строки экрана (системное окно), "S" - главная часть экрана и "P" - стандартный "ZX-принтер".

К этим каналам стандартно подключены потоки:

- поток #0 - к каналу "K";
- поток #1 - тоже подключен к каналу "K";
- поток #2 - подключен к каналу "S";
- поток #3 - к каналу "P".

Таким образом, оказываются идентичными следующие команды ввода/вывода:

```
PRINT #0 "Hello"; A$ - то же самое, что и INPUT "Hello"; A$
PRINT "Hello" - то же самое, что и PRINT #2 "Hello"
LPRINT "Hello" - то же самое, что и PRINT #3 "Hello"
```

Номер, стоящий после знака # в вышеприведенных примерах, является номером потока. Поскольку эти потоки подключены стандартно и переподключены быть не могут, мы программируем на БЕЙСИКе и используем операторы INPUT, PRINT, LPRINT без указания номера потока.

Это то, что касалось стандартных каналов и потоков, но они могут быть и нестандартными. Так, если Вы работаете в локальной сети, то сеть становится еще одним каналом, к которому Вы подключите поток.

Вы знаете, что "Спектрум" может в любой момент времени выполнять только одно дело. Например, либо он печатает на экране, либо на принтере. Одновременно выдавать информацию и туда и туда он не может, поэтому в любой момент времени задействован только один канал ввода/вывода и, соответственно, только один поток, связанный с ним. Этот канал и этот поток называются текущими. В большинстве случаев, когда Вы работаете с компьютером, текущим является канал "K", несколько реже - канал "S".

Программируя в машинном коде, переключаться с канала "K" на "S" (и наоборот) очень просто. Информацию о том, какой канал является текущим в данный момент, несет нулевой бит системной переменной TVFLAG (5C3CH - 23612). Когда он выключен (равен нулю), используется канал "S", а когда включен - "K".

Две небольшие процедуры продемонстрируют разницу в их использовании.

Демо_S

```
213C5C            LD HL,TVFLAG
3600              LD (HL),00      ; Выключили бит 0 систем-
                                  ; ной переменной TVFLAG.
3E2A      LOOP    LD A,42         ; Загрузили в аккумулятор
                                  ; число 42 (код символа
                                  ; "*".
D7                RST 10H         ; Выдали на печать по те-
                                  ; кущему каналу то, что
                                  ; находится в аккумулято-
                                  ; ре.
18FB              JR LOOP         ; Возврат для повтора.
```

Демо_K

```
213C5C            LD HL,TVFLAG
3601              LD (TVFLAG),01  ; Включили бит 0 систем-
                                  ; ной переменной TVFLAG.
3E2A      LOOP    LD A,42
D7                RST 10H         ; Печать символа "*".
18FB              JR LOOP         ; Возврат для повтора.
```

Не менее просто переключаться с каналов "S" или "K" на канал "P". Здесь тоже достаточно изменить один бит. Это первый бит системной переменной FLAGS (5C3BH - 23611). Он должен быть выключен для каналов "S" и "K", но включен для канала "P".

Демо_P

```
213B5C              LD HL,FLAGS
CBCE                SET 1,(FLAGS)   ; Включили бит 1 систем-
                                    ; ной переменной FLAGS.
0600                LD B,00         ; Обнуление счетчика
                                    ; (подготовка к печати
                                    ; 256-ти символов).
3E2A        LOOP    LD A,42
D7                  RST 10H         ; Печать символа "*".
10FB                DJNZ LOOP       ; Возврат для повтора,
                                    ; пока счетчик не достиг-
                                    ; нет нуля.
C9                  RET             ; Возврат в вызывающую
                                    ; программу.
```

Вы можете также изменить текущий канал, переключившись на другой поток. Это можно сделать вызовом процедуры ПЗУ, называющейся CHAN_OPEN и находящейся по адресу 1601H. Перед тем, как ее вызывать, следует в аккумуляторе установить номер желаемого потока.

```
3E02                LD A,02         ; Ввели номер желаемого
                                    ; потока.
CD0116              CALL 1601H      ; Сделали его текущим.
3E2A        LOOP    LD A,42
D7                  RST 10H
18FB                JR LOOP
```

Нам необходимо знать эти азы потому, что если мы используем для печати из машинного кода команду процессора RST 10H, то должны иметь в виду, что она выдает информацию ТОЛЬКО В ТЕКУЩИЙ КАНАЛ. Прежде, чем Вы дадите компьютеру команду, что бы Вы хотели, чтобы он напечатал, надо сначала определиться, куда он будет это печатать и как.

Команду RST 10H Вы можете использовать для печати любых символов, будь то символ стандарта ASCII или графический сим-

вол. Это может быть управляющий символ и даже токен ключевого слова стандартного БЕЙСИКа. Поместите код того, что хотите напечатать, в аккумулятор и дайте команду RST 10H. При этом обычные символы займут одно знакоместо, управляющие коды сделают то, что им положено, а токены ключевых слов будут развернуты и займут столько знакомест, сколько букв в этом ключевом слове. Вам надо также знать, что команда RST 10H никогда не портит содержимое регистров процессора, кроме BC', DE' (альтернативные) и аккумулятора, который портит не всегда.

2.2 ПЕЧАТЬ НА ЭКРАН ИЗ МАШИННОГО КОДА

2.2.1. Печать чисел.

1. Целые числа от 0 до 9. Выполнить печать целого числа от 0 до 9 можно двумя способами.

Во-первых, Вы можете напечатать его обычным способом, как и любой другой символ. Для этого поместите в аккумулятор процессора его код (код 0 - 48 (30H),... код 9 - 57 (39H)) и дайте команду RST 10H.

Во-вторых, можно это число напечатать, загрузив в аккумулятор не его код, а само число, но в этом случае печать следует выполнять не командой RST 10H, а вызовом специально для этого предназначенной процедуры ПЗУ - CALL 15EFH (десятиричный адрес - 5615), она называется OUT_CODE. Это, конечно, удобнее, но при своей работе эта процедура портит регистр E (имейте это в виду).

Возможен и промежуточный вариант, когда Вы засылаете в аккумулятор само число, а не его код, затем прибавляете к нему 30H (получаете его код) и затем даете RST 10H.

```
LD A,N
ADD A,30H
RST 10
```

По расходу памяти это то же самое, что и CALL OUT_CODE, но не портит регистр E.

2. Целые числа от 0 до 9999.

Для печати целых чисел, меньших чем 10000, введите это число в регистровую пару BC и вызовите процедуру ПЗУ OUT_NUM_1. Она находится по адресу 1A1BH (6683).

Если Ваше число содержится в виде двух байтов в известном Вам адресе памяти, то Вы можете воспользоваться процедурой ПЗУ OUT_NUM_2 (1A28H = 6696). В этом случае перед вызовом процедуры надо в регистровую пару HL заслать адрес, в котором находится Ваше число.

И в том и в другом случае, если Вы попробуете через эти процедуры распечатать число, которое больше 9999, результат будет неверным.

3. Целые числа от 0 до 65535.

В этом случае Ваше число тоже должно быть помещено в регистровую пару BC, но в отличие от предыдущего случая необходимо делать не один вызов процедур ПЗУ, а два.

Сначала оно должно быть конвертировано в интегральную (пятибайтную форму) и помещено на стек калькулятора. Это делается вызовом процедуры STACK_BC (CALL 2D2BH = 11563). И только после этого оно может быть напечатано, как действительное число с плавающей точкой. Это выполняется вызовом процедуры PRINT_FP, расположенной по адресу 2DE3H (11747).

4. Отрицательные целые числа.

Знак "минус" имеет код 2DH. Поместите его в аккумулятор, выполните RST 10H и абсолютную величину числа печатайте, как показано выше.

5. Произвольные действительные числа (числа с плавающей точкой).

Такое число занимает 5 байтов и хранить его ни в каком регистре, ни в регистровой паре невозможно. В этом случае Вами должны быть приняты меры для того, чтобы предварительно разместить его на вершине стека калькулятора. Когда это сделано, вызов процедуры PRINT_FP (2DE3H=11747) напечатает его на экране.

Здесь надо сделать пару предупреждений для тех, кто работает с калькулятором "Спектрума", "зашитым" в ПЗУ.

Во-первых, после работы PRINT_FP число со стека калькулятора снимается и, если Вам оно еще может потребоваться, то позаботьтесь предварительно продублировать вершину стека калькулятора (соответствующая команда в сводке команд калькулятора имеется - см. "Первые шаги в машинном коде", М.: "ИНФОРКОМ", 1990г.,1992г.).

Во-вторых, в процедурах ПЗУ, обслуживающих калькулятор, есть ошибка. Она заключается в том, что это число при вызове PRINT_FP снимается со стека не всегда. Если Ваше действительное число находится в диапазоне от -1 до +1 (ноль исключается), то вместо Вашего числа на вершине стека остается 0. Обращайте на это внимание.

2.2.2. Печать символьных строк.

У Вас есть по крайней мере три способа печатать текстовые сообщения, если Вы работаете в машинном коде. Но во всех случаях этот текст должен храниться в памяти компьютера и начинаться с известного Вам адреса.

Самый простой метод состоит в следующем. Регистровая пара DE должна содержать адрес, с которого начинается Ваша символьная строка, а в регистровой паре BC необходимо предварительно установить длину этой строки. Печать выполняется вызовом процедуры PR_STRING, которая находится по адресу 203CH (8252).

Во-вторых, символьная строка может быть Вами получена в результате работы встроенного калькулятора. В этом случае она

находится на вершине стека калькулятора и может быть напечатана прямо оттуда вызовом процедуры ПЗУ PR_STR_1 (адрес 2036H=8246).

Третья возможность, самая мощная, и именно она применяется в большинстве случаев в игровых, прикладных и системных программах.

У Вас может быть целый набор из N различных сообщений. И Вы, допустим, хотите напечатать k-ое сообщение. Тогда можете действовать следующим образом:

- установите в аккумуляторе число k-1;
- установите в регистровой паре DE адрес, указывающий на байт, находящийся перед первым байтом самого первого сообщения из Вашей таблицы сообщений. Имейте в виду, что этот байт должен быть больше или равен 80H, но меньше, чем FFH, т.е. старший (седьмой) бит в этом байте должен быть включен (он явится маркером начала текстового сообщения);

- вызовите процедуру PO_MSG (0C0AH=3082) и Ваше сообщение будет напечатано на экране.

Впрочем, у этого метода есть ряд ограничений и требований. Их необходимо также иметь в виду при программировании:

- недопустимо использование графических символов или токенов ключевых слов БЕЙСИКа, т.е. коды печатаемых символов должны быть менее 128 (80H). Впрочем, можно ведь и поменять символьный набор, заменив на время символы ASCII на нужные Вам графические шаблоны;

- во-вторых, Вам надо при заполнении таблицы сообщений сделать так, чтобы последеий символ каждого сообщения имел включенный 7-ой бит, тем самым он будет служить маркером конца i-го сообщения и компьютер всегда по номеру, заданному в аккумуляторе, найдет нужное Вам сообщение;

- в таблице не может быть пустых строк.

2.2.3. Полезные советы.

Работая с графикой из машинного кода, Вам надо иметь в виду некоторые особенности, связанные с работой встроенного калькулятора. Дело в том, что он не вполне свободен от разнообразных ошибок и неточностей, а они могут доставить головную боль начинающему программисту. Совсем другое дело, когда он предупрежден.

Итак, во-первых, если Вам необходимо выдать на экран символы блочной графики (коды с 80H по 8FH) - эти символы расположены на цифровом ряду клавиатуры, - то коррумпируются (портятся) системные переменные, расположенные в адресах с 5C92H (23698) по 5C99H (23705). Те, кто читал "Первые шаги в машинном коде", знают, что здесь хранятся две первые ячейки памяти встроенного калькулятора (из шести стандартных). Это ячейки M0 и M1. Почти во всех случаях Вам эта порча безразлична, но если Вы активно работаете с калькулятором и именно они Вам и нужны, то примите меры по сохранению их значений в другом месте, например на стеке калькулятора или в других ячейках.

Во-вторых, при печати чисел мы довольно активно пользовались процедурой ПЗУ PRINT_FP, которая распечатывает содержимое вершины стека калькулятора, а она во время своей работы "портит" все шесть ячеек памяти калькулятора, т.е. все содержимое системной переменной MEMBOT с адреса 5C92H (23698) по адрес 5CAFH (23727). Опять же для Вас это почти всегда безразлично, за исключением тех редких случаев, когда Вы оставили там на хранение данные без присмотра.

А вот третий случай довольно часто встречается у тех, кто много работает со встроенным калькулятором. Дело в том, что в таблице системных переменных есть переменная под названием MEM, она расположена в адресе 5C68H (23656) и занимает 2 байта. В ней хранится адрес, по которому расположена память калькулятора, т.е. адрес системной переменной MEMBOT. Если Вам достаточно тех шести ячеек памяти калькулятора, которые есть стандартно, то нет проблем. А если же Вам их мало, то Вы создаете их по-

больше, для чего меняете адрес MEMBOT и, естественно, указание на нее, содержащееся в MEM. Так вот, после такой замены процедура PRINT_FP правильно работать не сможет.

2.3 ИСПОЛЬЗОВАНИЕ УПРАВЛЯЮЩИХ КОДОВ

Набор символов "Спектрума" включает в себя стандартные символы ASCII - их коды с 20H (32) по 7FH (127), символы блочной графики, графики пользователя и токены ключевых слов БЕЙСИКа - коды с 80H (128) по FFH (255). Это оставляет вне поля нашего зрения символы с 0 по 1FH (31). Что же расположено там?

А здесь расположены так называемые управляюшие символы, их еще называют управляющими кодами (символами их называть и не хочется, ведь их нельзя напечатать и увидеть). Хоть сами они на экране и не воспроизводятся, зато с их помощью можно управлять процессом печати тех символов, которые могут быть напечатаны.

Мы начнем с управляющего кода 16H. Он называется AT CONTROL и определяет координаты позиции печати так же, как и оператор AT в БЕЙСИКе.

Задействуются управляющие коды, как и любые другие символы путем выставления этого кода в аккумуляторе процессора и подачей команды RST 10H. Это означает, что они могут быть как бы "напечатаны" вместе с прочими символами и могут быть включены в длинные символьные строки.

БЕЙСИКовский оператор AT требует после себя указания двух параметров - координат позиции печати по вертикали и горизонтали. Управляющий код AT требует то же самое. Оба параметра вводятся тем же способом - вводом параметра в аккумулятор и выдачей команды RST 10H.

Приведенная распечатка показывает машиннокодовый аналог оператора БЕЙСИКа PRINT AT 5,4; .

```
                DEMO AT_5_4

3E16        LD A,16H
D7          RST 10H     ; PRINT AT ...
3E05        LD A,05
D7          RST 10H     ; 5,.....
3E04        LD A,04
D7          RST 10H     ; 4, ....
```

Не имеет никакого значения сколько и каких машиннокодовых команд будет выдано между тремя вышеприведенными командами RST 10H. "Спектрум", если выдал управляющий код 16H, далее всегда будет помнить, что два ближайших числа, проходящих через RST 10H являются параметрами управляющего кода AT_CONTOL.

Более часто этот управляющий код находит применение, когда речь идет не о печати в фиксированной позиции, а в позиции, заданной программными переменными. Ниже показан машиннокодовый аналог команды PRINT AT D,E (здесь D и E - содержимое соответствующих регистров процессора.)

```
                DEMO AT_D_E

3E16        LD A,16H
D7          RST 10H     ; PRINT AT ...
7A          LD A,D
D7          RST 10H     ; D,.....
7B          LD A,E
D7          RST 10H     ; E, ....
```

Следующий управляющий код, который мы рассмотрим - это код 17H. Он называется TAB_CONTROL - код управления табуляцией. До некоторой степени он тоже аналогичен оператору БЕЙСИКа TAB и указывает на номер позиции печати в текущей строке.

За ним также следуют 2 параметра (это не ошибка - именно 2 параметра, хотя БЕЙСИКовский опыт учит, что для позиции печати достаточно и одного). Дело в том, что первый параметр - это

младший байт координаты позиции печати, а второй параметр - ее старший байт. Поскольку экран "Спектрума" имеет всего лишь 32 символа по ширине, сразу и не понятно зачем еще нужен какой-то старший байт? Но дело в том, что печать ведь может идти не только на экран. А если на принтер? Да, ZX-принтер тоже имеет только 32 символа в строке, но ведь возможна печать и на широкий матричный принтер, через интерфейс. Внимательный читатель может спросить, где мы видели принтер, печатающий в строку более 255 символов, но это уже вопрос риторический. Неважно, есть он или нет, но возможность работы с ним в компьютере предусмотрена. Более того, понятие "печать", как способ выдачи информации, может предусматривать не только стандартные каналы типа экрана или принтера. Возможны ведь и пользовательские каналы типа файлов на диске или в оперативной памяти, в которых запись может иметь и десятки тысяч символов в строке.

Но мы имеем дело все же с экраном, поскольку речь идет о графике, и для печати на экране то, что находится в старшем байте, не имеет никакого значения. Дело в том, что когда Вы задаете параметр TAB, компьютер берет из него остаток от деления на 32, а старший байт кратен 256 и, естественно, кратен 32, поэтому его содержимое влияние при печати на экран и не оказывает.

Нижеприведенная процедура показывает управляющий код TAB CONTROL в действии. Первый параметр предполагается переменным и берется из регистра E. Выдавая второй параметр, мы обнулили его, использовав для этого обнуление аккумулятора через XOR A, но раз этот байт роли не играет, можно было XOR A и не давать, а отправить в этот параметр то, что было в аккумуляторе к этому моменту, неважно, что это было.

```
                DEMO TAB_E
3E17        LD A,17
D7          RST 10H          ; PRINT TAB...
7B          LD A,E
D7          RST 10H          ; E, ...
AF          XOR A
D7          RST 10H          ; 0 ...
```

Код CHR 6 называется COMMA_CONTROL и выполняет то же, что и запятая оператора PRINT. Код подается точно так же, как и прочие, введением числа 6 в аккумулятор и выдачей его через RST 10H. В результате его действия в качестве позиции печати устанавливается 0 или 16, в зависимости от того, в какой половине экрана печаталось последнее знакоместо.

Код 0DH (13) - аналогичен ENTER. Выдается так же и обеспечивает прекращение печати в текушей строке и переход в начало следующей.

Код 08 - называется BACKSPACE_CONTROL. Он выполняет смещение позиции печати на одно знакоместо влево.

Команда процессора RST 10H - довольно гибкая и имеет самое разнообразное действие. С ее помощью можно не только управлять позицией печати символов на экране, но и управлять цветами того, что Вы воспроизводите, т.е. выдавать коды цветовых атрибутов. Выполняется это точно так же путем предварительной установки в аккумуляторе управляющего кода и передачей его, а затем передачей параметра. Мы не будем подробно останавливаться на цветовых атрибутах и привели их в Таблице 4.

Пример применения кодов управления цветовыми атрибутами показывает, как выполняется печать красной звездочки на желтом фоне. Обратите внимание на то, что установленные таким способом цветовые атрибуты остаются действующими до того, как будут изменены или до окончания действия оператора БЕЙСИКа, из которого вызывалась данная процедура в машинных кодах.

```
3E10            LD A,10H
D7              RST 10H     ; PRINT INK...
3E02            LD A,02
D7              RST 10H     ; 2;....
3E11            LD A,11H
D7              RST 10H     ; PAPER...
3E06            LD A,06
D7              RST 10H     ; 6;....
3E2A            LD A,2A
D7              RST 10H     ; "*";
```

Таблица 4.

Управляющий код	Параметры	Комментарии
06 COMMA_CONTR 08 BACKSPACE 0D ENTER	- - -	
10 INK_CONTROL 11 PAPER_CONTR	00-черный 01-синий 02-красный 03-пурпурн. 04-зеленый 05-голубой 06-желтый 07-белый 08-транспарантный 09-контрастный	Параметр 08 (транспарантный) означает, что цвет печати не устанавливается, а берется тот, который уже есть в данной позиции печати. Параметр 09 (контрастный) устанавливает цвет INK или PAPER контрастным к уже существующему в данной позиции противоположному цвету.
12 FLASH_CONTROL 13 BRIGHT_CONTR	00-выключен 01-включен 08-транспарантный	См. выше.
14 INVERSE_CONTR 15 OVER_CONTROL	00-выключен 01-включен	
16 AT_CONTROL 17 TAB_CONTROL	Строка, столбец. Младший байт, старший байт.	При выдаче на экран старший байт может быть любым. Можете, например использовать эту ячейку памяти для хранения какой-либо однобайтной переменной.

2.4. ДРУГИЕ ПРИЕМЫ УПРАВЛЕНИЯ ПЕЧАТЬЮ

В "Спектруме" есть возможность задействовать некоторые процедуры ПЗУ и управлять системными переменными для того, чтобы достигать тех же эффектов, которые дает применение управляющих кодов. Рассмотрим эти приемы.

PRINT AT B,C (где B и C -содержимое одноименных регистров процессора) может быть выполнено вызовом процедуры AT_B_C, находящейся по адресу 0A9BH (2715).

PRINT TAB A (где A - содержимое аккумулятора) выполнимо вызовом процедуры TAB_A (0AC3H=2755).

Управляющий код 06 (COMMA_CONTROL) эмулируется вызовом PO_COMMA (0A5FH=2655).

Эти три приема действуют не только на канал "S", но и на канал "K", т.е. с их помощью можно печатать информацию и в INPUT-строке.

У Вас есть также достаточно простой способ управлять цветовыми атрибутами INK, PAPER, BRIGHT, FLASH. Все, что для этого требуется - внести изменения в системную переменную ATTR_T (5C8FH = 23695), отвечающую за статус временных атрибутов экрана. Эта системная переменная имеет следующую раскладку:

Биты 0...2 - цвет INK (от 0 до 7)
Биты 3...5 - цвет PAPER (от 0 до 7)
Бит 6 - статус BRIGHT (0 или 1)
Бит 7 - статус FLASH (0 или 1)

Все, что требуется для установки нужных комбинаций цвета - это заслать в эту системную переменную число, определяемое по формуле:

128*F + 64*B + 8*P + I, где:

- F - статус FLASH;
- B - статус BRIGHT;
- P - номер цвета PAPER;
- I - номер цвета INK.

Если же Вы хотите, чтобы какие-то цветовые атрибуты были

транспарантными, системной переменной ATTR_T Вам уже недостаточно и надо воспользоваться системной переменной MASK_T (5C90H = 23696). Ее раскладка точно та же, что и у ATTR_T. Биты, соответствующие атрибуту, который Вы хотите сделать транспарантным, надо включить.

Итак, рассмотрев, каким образом из машинного кода выполняется печать графики низкого разрешения, мы переходим к графике высокого разрешения, но для этого надо сначала хорошо разобраться со структурой экрана и, главное, понять как раскладка экрана связана с организацией экранной памяти. Отсюда мы сделаем шаг к наиболее рациональным приемам по изменению экранных форм из машинного кода.

2.5. ОРГАНИЗАЦИЯ ЭКРАННОЙ ПАМЯТИ

Область оперативной памяти "Спектрума", в которой находится графическое изображение, видимое на экране в качестве включенных и выключенных точек (пикселов), называется экранной областью памяти.

Эта область занимает адреса с 4000H по 5AFFH (с 16384 по 23295). Причем, она состоит из двух областей. В первой с 4000H по 57FFH (с 16384 по 22527 расположено растровое изображение, состоящее из черных и белых неокрашенных точек (дисплейный файл). Во второй, с 5800H по 5AFFH (с 22528 по 23295) - цветовая информация (файл атрибутов).

Размер всей экранной области - 6912 байтов. Из них 6144 байта - дисплейный файл (черно-белый) и 768 байтов - файл атрибутов (цвет).

Как получены эти числа 6144 и 728?

Вы знаете, что полный экран "Спектрума" может иметь 24 ряда по 32 символа в ряду, т.е. всего 24*32=728 знакомест. Каждое знакоместо образовано из 64-х пикселов (8 линий по 8 точек). На каждую линию достаточно одного байта (8 битов соответствуют 8 точкам) и, следовательно, для растровой графики одного знакоместа необходимы 8 байтов. Так, на 728 знакомест необходимо 728*8=6144 байта для хранения монохромной информации.

Информация о цвете хранится более экономно. Каждому знакоместу отдан один байт, определяющий окраску всех 64 его пикселов. Три младших бита определяют цвет INK этого знакоместа, еще три бита определяют цвет PAPER и по одному биту отдано признакам BRIGHT и FLASH. Отсюда вытекает одно из самых неприятных ограничений графики "Спектрума" - в пределах одного знакоместа невозможно иметь одновременно более 2-х цветов.

А теперь давайте немного поэкспериментируем. Очистите экран - CLS. Окрасьте бордюр в голубой цвет - BORDER 5. Это

нужно, чтобы точно видеть результат следующих манипуляций и дайте команду, задающую самый первый байт экранной области:

```
POKE 16384,255
POKE 16384,15
POKE 16384,85
```

После каждой команды Вы увидите изменения в левом верхнем углу экрана, а точнее - в первой линии первого знакоместа. Вам должно быть понятно, что и почему там происходит. Вы видите, что раскладка пикселов в этой линии соответствует раскладке битов в байте 16384, отвечающем за эту линию.

Теперь попробуем закрасить вторую линию в первом знакоместе. Логично предположить, что POKE в следующую ячейку памяти сдедает это. Дайте POKE 16385,85. Получилось совсем не то - включилась первая линия второго знакоместа. Далее - третьего и так далее, до 32-го. Может быть, попробуем их пропустить и дадим POKE сразу в 33-ий адрес:

```
POKE (16384+32),85
```

И опять ничего не получилось. Вместо второй линии в первом знакоместе включается первая линия в первом знакоместе, но во втором ряду. И так будет продожаться, пока Вы не пройдете все 32 знакоместа в 8 рядах экрана, т.е. 32*8=256 адресов. Только дав POKE (16384+256), Вы получите то, что хотели. На первый взгляд такое соответствие между точками экрана и ячейками экранной памяти выглядит достаточно нелепым и нелогичным. Но это не совсем так. Как окажется чуть ниже, это не только не усложняет жизнь, а при работе из машинного кода даже упрощает - надо только хорошо все понять.

Теперь, покопавшись в памяти, Вы наверняка вспомните, что уже много раз видели при загрузке заставок игровых программ, как экран прорисовывается постепенно строчка за строчкой. В этот момент Вы и видели соответствие между структурой экрана и экранной памятью. Хотите повторить, пожалуйста:

```
10 CLS: BORDER 5
20 FOR i=16384 TO (16384+256*8)
30 POKE i,255
40 NEXT i
```

Чтобы не утомлять Вас длительным ожиданием, мы сделали это только для 8 первых рядов экрана.

Научившись изображать на экране точки и тире без помощи операторов PRINT и PLOT, мы уже сделали большое дело и развязали себе руки, т,к. мы теперь умеем манипулируя с ячейками памяти делать графику, а машинный код именно и хорош, когда дело доходит до манипуляций с большими объемами памяти.

Но как насчет того, чтобы получить нормальный графический образ? Нет проблем, этот образ сначала надо где-то в памяти создать. Давайте воспользуемся готовым. Вы, конечно знаете, что в ПЗУ компьютера где-то имеется набор символов, в котором хранятся графические образы (шаблоны) всех букв и прочих знаков. Вот мы возьмем оттуда образ буквы "A" и нарисуем ее на экране без PLOT или PRINT.

В адресах 23606, 23607 (5C36H) хранится 2-х байтная системная переменная CHARS, которая указывает на 256 байтов ниже, чем адрес, с которого начинается набор символов. Во-первых, давайте разберемся, почему она указывает ниже на 256 байтов, а не туда, куда надо. Все очень просто. Мы уже говорили о том, что первые 32 символа вовсе и не символы, а управляющие коды и им графические образы не нужны, все равно они не печатаются. А т.к. символы занимают по 8 байтов каждый, то на пропуск этих 32 управляющих кодов и ушло это снижение на 256 байтов. Зато теперь мы можем искать образ буквы A по ее номеру (по ее коду ASCII, который, кстати, равен 65(41H)).

```
10 LET base=PEEK 23606 + 256*PEEK 23607
20 LET addr = base+8*65
25 LET screen=16384
30 FOR i=0 TO 7
```

```
40 LET pic=PEEK (addr+i)
50 POKE (screen+256*i),pic
60 NEXT i
```

Мы с Вами только что нарисовали букву "A", причем именно нарисовали, а не напечатали, да к тому же работали при этом только с переброской данных из одних участков памяти в другие. Вы обратили внимание на то, что засылали мы графику в дисплейный файл в строке 50 с шагом через 256 адресов? Давайте теперь рассмотрим, как то же самое будет выглядеть в машинном коде и почему так сложно, на первый взгляд, организован дисплейный файл.

Адреса экранной памяти обычно при работе с экраном хранятся в регистровой паре HL. При этом в H хранится старший байт адреса (High - старший), а в L - младший (Low). Для того, чтобы увеличить адрес на 256 и перейти к нижележащей линии в том же знакоместе оказывается достаточно увеличить на единицу старший байт. С этой задачей изящно справляется простая команда АССЕМБЛЕРа INC H. Увеличение же на единицу младшего байта адреса (INC L) переместит Вас на соседнее знакоместо вправо в той же линии. Видите, как все просто. И преобразование вышеприведенной БЕЙСИК-программы в машинный код выглядит так:

```
LD DE,(23606)       ; Загрузили в DE содержимое
                    ; системной переменной CHARS.
LD HL,0041H         ; Загрузили в HL код буквы A.
ADD HL,HL           ; Умножили его на 2.
ADD HL,HL           ; Умножили его на 4.
ADD HL,HL           ; Умножили его на 8.
ADD HL,DE           ; Теперь HL указывает на начало
                    ; шаблона буквы A.
EX DE,HL            ; Освободили HL для работы с
                    ; экраном.
LD HL,4000H         ; Загрузили в HL адрес начала
                    ; экранного файла.
LD B,08             ; Организуем счетчик на 8
                    ; шагов.
```

```
LOOP    LD A,(DE)          ; Переброска содержимого из
        LD (HL),A          ; DE в экранный файл через
                           ; аккумулятор.
        INC H              ; Переход к новой линии экрана.
        INC DE             ; Переход к новой линии шаблона.
        DJNZ LOOP          ; Окончание цикла из 8-ми шагов.
        RET                ; Возврат.
```

Итак, все вроде бы просто и понятно, но Вы, конечно, обратили внимание на то, что все, что мы до сих пор делали, относится только к восьми первым символьным рядам экрана, а что же дальше? Ведь их всего 24.

Здесь тоже все просто, если представить себе, что экран состоит из трех несвязанных между собой областей, каждая из которых имеет по 8 рядов. Назовем эти трети экрана СЕГМЕНТАМи. Итак, экран состоит из трех сегментов, каждый из которых состоит из восьми рядов, каждый из которых состоит из тридцати двух знакомест, каждое из которых состоит из восьми линий, каждая из которых состоит из восьми пикселов.

Первый сегмент - 16384 - 18431 (4000H - 47FFH)
Второй сегмент - 18432 - 20479 (4800H - 4FFFH)
Третий сегмент - 20480 - 22527 (5000H - 57FFH)
Каждый сегмент занимает по 8*32*8=2048 байтов.

Если теперь развернуть регистровую пару HL виде шестнадцати битов, то получится любопытная побитовая карта для дисплейного файла (Рис. 11).

Однажды, в 1985 году сэра К.Синклера спросили, чем он объясняет столь невероятный успех своих компьютеров по сравнению с главными конкуретами "Коммодором", "Атари", "Амстрадом" и "Би-Би-Си Микро", если известно, что графика у них лучше, музыка богаче, надежность выше и дополнительных внешних портов больше? На это он ответил: "У меня гораздо проще доступ к памяти". И этим сказано все. В частности, организация экранной памяти в "Спектруме" была не последним моментом, обеспечившим

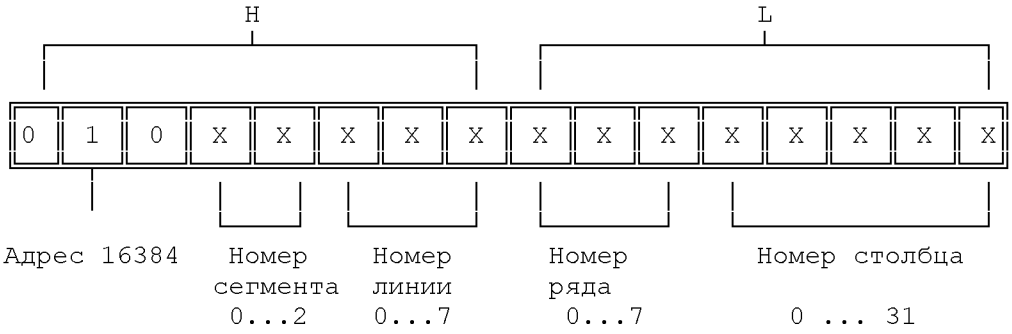


Рис. 11

ему поддержку со стороны сотен фирм, выпускающих программное обеспечение.

Давайте рассмотрим эти преимущества.

1. В отличие от многих других разработчиков К.Синклер нашел удачное решение, объединив в едином экране и графику высокого разрешения и символьную графику. Нет необходимости переключаться на другой экран всякий раз, как надо построить диаграмму или написать текст. Это большой подарок программистам.

2. Решение о разделении черно-белой информации высокого разрешения и цветовой информации низкого разрешения в разные файлы позволило ему в 4-5 раз уменьшить объем экранной памяти и выделить много места для пользователя. Практически здесь имитируется цветная графика высокого разрешения, хотя она таковой строго говоря и не является - это был гениальный ход, высоко оцененный специалистами.

3. Эти решения позволили в значительной степени сократить размеры программ ПЗУ. Действительно, по отношению возможностей ПЗУ к его размерам вряд ли найдется машина, равная "Спектруму", хотя и у этого ПЗУ, как показал дальнейший опыт, еще были огромные резервы.

Теперь посмотрим, как на практике ухватились программисты например за такой простой элемент, как сегментирование экрана на три зоны при том, что экран является одновременно и графическим и символьным.

Во-первых, множество первоклассных программ используют для своей графики только один сегмент, оставив прочее на диалог с пользователем. Ведь стоит только вдуматься, что в таких программах, как TIR-NA-NOG, DUN DARACH, MARSPORT фирмы "GARGOYLE GAMES" огромные события происходят на участке памяти размером всего лишь 2К.

Во-вторых, стал традиционным прием, при котором графика занимает вроде бы весь экран и непрофессионал даже и не замечает, что два сегмента из трех заняты красочной статической графикой, в то время как все действие разворачивается лишь на одном сегменте (может быть и менее красочном). Малый размер памяти этого сегмента позволяет достичь высокого быстродействия, плавности перемещения объектов, прекрасной реакции на действия пользователя, а общий эстетический эффект непроизвольно распространяется на весь экран. Вроде бы имитация, но какая!

В третьих, очень часто экранную память неиспользуемых сегментов начинают использовать для разных вспомогательных действий, для размещения временных таблиц, буферов и т.п., втискивая в небольшие размеры оперативной памяти огромное количество информации. Вы сами, возможно видели во время работы копировщика COPY-86M, как временно ненужная информация выбрасывается на экран в виде точек и тире в двух нижних сегментах экрана. Это применяют и в игровых программах, но там Вы этого не увидите, т.к. программист заблаговременно установил

в цветовых атрибутах, относящихся к этим сегментам экрана, черный цвет INK и черный цвет PAPER. За черным цветом прячут иногда даже и машинный стек процессора (например в программе NETHEREARTH), что не только экономит оперативную память, но является еще и надежным приемом защиты программы от внешнего вторжения, и многое-многое другое.

Файл атрибутов.

Теперь, рассмотрев работу с дисплейным файлом, перейдем к файлу атрибутов и посмотрим работу следующей программы:

```
10 PAPER 6: INK 0: BORDER 6: CLS
20 FOR i=1 TO 22
30 PRINT, "ZX-SPECTRUM"
40 NEXT i
50 FOR i=22528 TO 23295
60 POKE i,15
70 NEXT i
```

Посмотрите, что произойдет, когда Вы запустите эту программу. Во-первых, на экране будет напечатан текст (черным цветом по желтому фону). Во-вторых, знакоместо за знакоместом, начнется изменение его цвета (белые буквы по синему фону). Чтобы понять почему так происходит, нам надо изучить область памяти от адреса 22528 до адреса 23295 (5800H - 5AFFH), которая и называется файлом атрибутов. Содержимое ячеек памяти в этой области определяет то, каким цветом будет окрашено то или иное знакоместо.

Структура этого файла очень проста. Каждому знакоместу экрана соответствует один байт памяти, а значение этого байта и определяет цвета этого знакоместа. Это означает, что когда Вы засылаете (POKE) какое-либо число в ячейку памяти этой области, Вы изменяете цвет одного из знакомест экрана. Каждый байт из файла атрибутов хранит информацию о параметрах INK, PAPER, BRIGHT и FLASH.

Биты 0...2 - цвет INK (от 0 до 7)
Биты 3...5 - цвет PAPER (от 0 до 7)
Бит 6 - статус BRIGHT (0 или 1)
Бит 7 - статус FLASH (0 или 1)

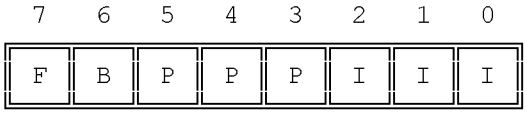


Рис. 12

Для программирующего в машинном коде, конечно, чрезвычайно важно знать точно, какому знакоместу экрана соответствует какой байт файла атрибутов. Зная, что в каждом ряду 32 знакоместа, этот адрес вычислить нетрудно: 22528 + 32*Y + X.

Здесь Y и X - координаты знакоместа. Y - номер ряда сверху вниз, X - номер столбца слева направо. Как видите, в отличие от дисплейного файла файл атрибутов организован вполне благоразумно : слева направо и сверху вниз.

Если теперь развернуть регистровую пару HL виде шестнадцати битов, то побитовая карта для файла атрибутов имеет следующий вид:

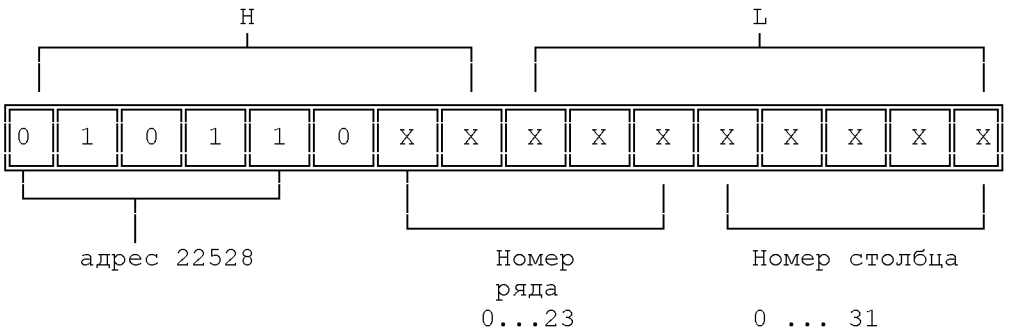


Рис. 13

2.6 УПРАВЛЕНИЕ ЦВЕТОМ БОРДЮРА

При работе в машинном коде здесь есть небольшая сложность, заключающаяся в том, что для изменения цвета бордюра требуется два действия, т.к. к этой цели ведут два пути.

Дело в том, что существует текущий цвет бордюра (тот, который Вы в данный момент видите на экране) и есть установленный цвет бордюра (тот, который записан в памяти компьютера и будет установлен, как только ПЗУ компьютера перехватит управление от Вашей программы в машинных кодах, например при нажатии какой-либо клавиши).

Ваша задача состоит в изменении обоих цветов - и текущего и установленного, т.к. изменение только текущего будет иметь кратковременный характер, а изменение только установленного вообще никак не отразится на экране в данный момент.

Текущий цвет бордюра изменяется выдачей байта по 254-ому внешнему порту (бордюрная часть экрана является для "Спектрума" внешним устройством, подключенным к порту FEH). Это можно сделать даже из БЕЙСИКа, воспользовавшись командой OUT:

```
OUT 254,цвет
```

В машинном коде эта команда выглядит удивительно похоже:

```
LD  A,цвет
OUT (FEH),A
```

Установленный цвет бордюра, с которым работает ПЗУ, постоянно хранится в отведенной для него ячейке памяти в области системных переменных. Ее название - BORDCR. Ее адрес - 23624 (5C48H). Три бита этой переменной отвечают за цвет бордюра:

Прочие биты занимаются другими делами, в частности, они задают цвет нижней части экрана, используемой для работы INPUT и пр.

Оба переключения одновременно можно сделать вызовом процедуры ПЗУ BORDER_A, расположенной по адресу 2297H (8855). Предварительно код цвета надо установить в аккумуляторе процессора. У нее есть и побочное воздействие. Если при вызове этой процедуры что-то содержится в системных нижних двух строках компьютера, то она обойдется с ними достаточно "гуманно". Чтобы бордюр не скрыл информацию, цвет INK этих строк будет установлен контрастным к цвету бордюра.

2.7. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ 128-КИЛОБАЙТНЫХ МАШИН

"Спектрумы" со 128 килобайтной памятью имеют не одну, а две экранных области. (См. "ZX-РЕВЮ", N1,с.4; N2, с.26; М.: ИНФОРКОМ, 1991), каждая из которых может быть использована для хранения изображения, если Вы работаете в режиме 128К.

Первая экранная область, как обычно занимает адреса 4000H - 5AFFH, а вторая расположена в адресах C000H - DAFF на седьмой странице оперативной памяти. Первая область называется нулевым экраном, а вторая - первым экраном. Очевидно, что только одна область из двух может быть воспроизведена на экране телевизора в данный момент.

Раскладка экрана-1 точно такая же, как и экрана-0 и мы можем считать, что разница состоит во-первых в 15-ом бите, а во-вторых в том, что он расположен не на нулевой странице ОЗУ, а на седьмой.

Если развернуть регистровую пару HL в виде шестнадцати битов, то получится побитовая карта для дисплейных файлов, приведенная ниже на Рис. 14.

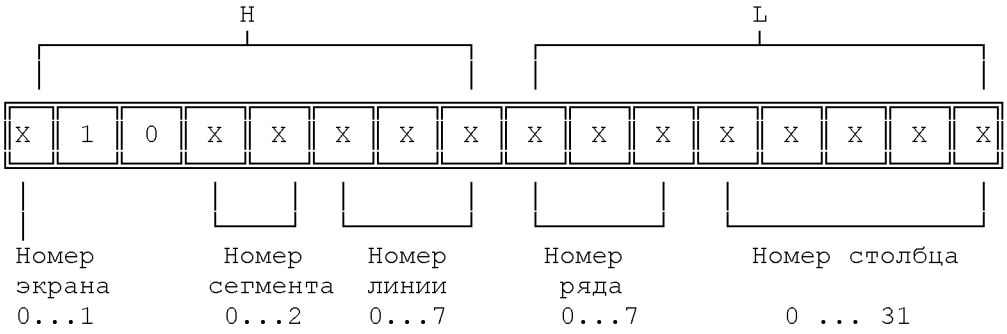


Рис. 14

Тот экран, который в настоящее время задействован, называется активным. Обратите внимание на интересную особенность первого (дополнительного) экрана. После того, как он был задействован, уже нет необходимости седьмой странице ОЗУ оставаться "впечатанной". Т.е. экран-1 может оставаться активным независимо от того, какие страницы ОЗУ "впечатаны" в данный момент.

Есть еще одна особенность: процедуры ПЗУ могут работать только с нулевым (основным) экраном. Они не умеют печатать и рисовать на втором экране. Ни PLOT ни PRINT, ни DRAW ни даже машиннокодовая команда RST 10H не работают с первым экраном. С ним не работают ни автоматический листинг, ни система редактирования, ни INPUT.

"Спектрум-128", как это ни странно, не содержит в своем ПЗУ-128 совершенно ничего, чтобы переключаться с одного экрана на другой и пользователю приходится самому писать для этого процедуру, например такую, как показано ниже. Точками входа в нее являются метки SCR_0 и SCR_1, соответствующие тому, какой экран Вы хотите вызвать.

```
1E00   SCR_0        LD E,00      ; Все биты выключены.
1802                JR SELECT    ; Переход на переключение
                                   экрана.
```

```
1E08    SCR_1       LD E,08       ; Включен бит-3.
3A5C5B  SELECT      LD A,(BANK_M) ; Системная переменная
                                  ; BANK_M (5B5CH) - служит
                                  ; в 128-килобайтных маши-
                                  ; нах как указатель стра-
                                  ; ниц ОЗУ, ПЗУ, экрана и
                                  ; режима (см. рис. 15).
                                  ; За номер экрана отвеча-
                                  ; ет бит 3.
E6F7                AND F7        ; Включили все биты в ак-
                                  ; кумуляторе, кроме 3-го.
B3                  OR E          ; Третий бит включается
                                  ; или выключается в зави-
                                  ; симости от того, через
                                  ; какую точку мы вошли в
                                  ; эту процедуру.
325C5B              LD (BANK_M),A ; Переключили BANK_M на
                                  ; экран-1. Но он еще не
                                  ; активен. Это только под-
                                  ; готовка.
01FDF7              LD BC,7FFD    ; Установили в BC номер
                                  ; внешнего порта, служа-
                                  ; щего для физического
                                  ; переключения страниц и
                                  ; режимов.
ED79                OUT (C),A     ; Переключение экрана.
C9                  RET           ; Выход
```

Примечание: Обязательный порядок всех переключений в 128-х машинах - сначала перенастраиваем BANK_M и только потом выдаем команду по порту 7FFD. Системные прерывания компьютера вызывают запуск его процедур ПЗУ, которые, заканчивая работу, выдают по порту 7FFD содержимое BANK_M. Так что, если Вы будете делать наоборот, есть высокая степень вероятности, что у Вас ничего не получится. Правда, из машинного кода прерывания можно и отключать (см. "Первые шаги в машинном коде").

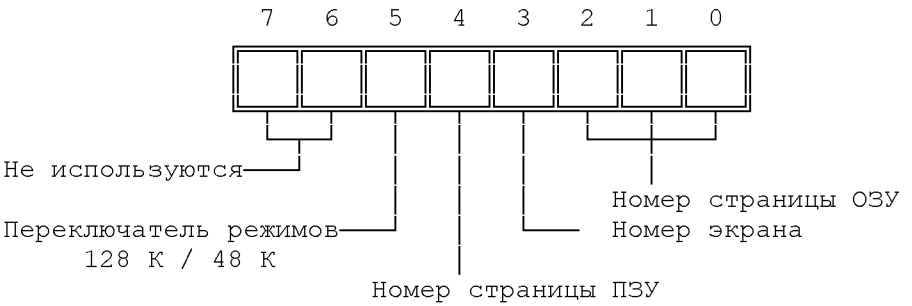


Рис. 15 Системная переменная BANK_M (5B5CH - 23388).

И еще одно предостережение для тех, кто работает с RAM-диском на 128-килобайтных машинах (о RAM-диске см. "ZX-РЕВЮ", N12,с.235-240; М.:ИНФОРКОМ, 1991.). Когда Вы выгружаете что-либо в RAM-диск, используя для этого новую команду SAVE!, файлы сохраняются в памяти в виде стека - один над другим. Начало области - 1C000 и далее вверх до 1FFFF, потом с 3C000 по 3FFFF, с 4C000 по 4FFFF, с 6C000 по 6FFFF и, наконец, с 7C000 и вверх.

Одновременно с этим образуется второй стек. Он начинается в адресе 7EBFF и развивается вниз. В нем содержатся только имена и адреса файлов, выгруженных на RAM-диск. Короче говоря, этот стек содержит ту информацию, которую Вы получаете, воспользовавшись новой командой CAT! . Таким образом, эти два стека развиваются навстречу друг другу. Встречаться они не должны. При появлении такой опасности компьютер выдаст сообщение об ошибке "4 Out of memory". Стек каталога не может также опуститься ниже 7C000 - именно поэтому установлено ограничение на количество сохраняемых файлов - 562.

Итак, все сделано, чтобы эти стеки не встретились и не испортили друг друга, но ничего не сделано для того, чтобы они не затерли экран-1. Этот экран может быть испорчен как стеком

файлов, так и каталожным стеком. И даже наоборот, работая с экраном-1, Вы можете испортить эти стеки.

Будьте внимательны. Работая в машинном коде, использовать экран-1 можно только если Вы не очень активно используете виртуальный диск. Смело можете хранить до 216 файлов с суммарной длиной не более 64К.

2.8. ЭМУЛЯЦИЯ КОМАНД БЕЙСИКА ИЗ МАШИННОГО КОДА

Здесь мы рассмотрим как выполнить БЕЙСИК-овские команды, связанные с графикой, из машинного кода. Как правило это несложная задача, поскольку в ПЗУ имеются процедуры, способные это сделать, надо только знать, каким образом к ним следует обращаться.

2.8.1 CLS (CLEAR SCREEN)

Сначала займемся очисткой экрана. В машинном коде это можно сделать вызовом процедуры CLS, расположенной в ПЗУ по адресу 0D6BH (3435). В результате ее работы выключаются все пикселы на экране, а цветовые атрибуты устанавливаются такими, как установлено в системной переменной ATTR_P 5C8DH (23693). Ее побитовая раскладка стандартна для атрибутов (см. Рис.8, с.16) Данная процедура работает точно так же, как и соответствующая команда БЕЙСИКа.

Если хотите, то можете очистить только нижнюю часть экрана (обычно это две нижние строки). Это делается вызовом процедуры CLS_LOWER, которая находится по адресу 0D6EH (3438). В отличие от основного экрана эта область очищается с цветом бордюра, а не с цветом, установленным в ATTR_P (см. с. 72) .

Среди процедур ПЗУ есть еще одна, обеспечивающая более мощные возможности. Она называется CL_LINE (0E44H = 3652) и очищает заданное количество строк в нижней части экрана. Перед ее вызовом в регистре B должно быть установлено количество строк, подлежащих очистке от 01 до 18H (от 1 до 24). Так, например, чтобы очистить 10 нижних строк, нужно предварительно в регистр B заслать число 0AH.

Эту процедуру можно применять двумя способами. Если нулевой бит системной переменной TVFLAG (5C3C = 23612) выключен, то при очистке используются цвета экрана, а если он включен, то используется цвет бордюра. Таким образом, здесь есть возмож-

ность очистить одновременно весь экран, включая и нижние две строки, в цвета ATTR_P. Процедура CLS сделать такого не может. В то же время, следует обратить внимание на то, что процедура CLS после очистки экрана инициализирует курсор и текущая позиция печати устанавливается в исходное положение - левый верхний угол. Процедура же CL_LINE этого не делает.

2.8.2 Скроллинг экрана.

Когда Вы пытаетесь напечатать что-либо за последней возможной позицией печати, то вместо результата получаете сообщение "scroll?" в нижней части экрана и компьютер ждет от Вас нажатие клавиши. Если это клавиша "N" или "BREAK", печать на экране прекращается с сообщением об остановке, в противном случае печать продолжается. Экран вроде бы заполнен, но печать возможна, происходит скроллинг вверх. Этот скроллинг происходит автоматически и длится до тех пор, пока вся информация, присутствовавшая на экране, когда выходило сообщение "scroll?", не скроется из виду за верхней кромкой экрана.

Такой вид скроллинга называется автоматическим и он существует на "Спектруме" как в БЕЙСИКе, так и в машинном коде.

Но возможен также и "ручной" скроллинг. Вызов процедуры CL_SC_ALL (0DFEH = 3582) "прокрутит" весь экран, но при этом возникнут несколько неожиданные эффекты:

- во-первых, позиция печати не изменится и останется там же, где была. С этим Вам придется разбираться вручную.

- во-вторых, если первая строка нижней части экрана (системного окна) не пуста или имеет цвет, отличный от цвета основного экрана, то результат может быть не сосвсем тем, какой Вы ожидаете.

Другой способ исполнения "ручного" скроллинга еще интереснее. Он базируется на использовании процедуры CL_SCROLL

(0E00H = 3584). С ее помощью можно "прокручивать" только часть экрана. Предварительно в регистр B надо занести количество строк, подлежащих скроллингу (обратите внимание на то, что изменено будет на одну строку больше). Минимальное количество строк - 1. В этом случае нижняя строка экрана будет поднята на одно знакоместо вверх, а снизу появится чистая строка. Для того, чтобы переработать весь экран, следует установить число 17H. Таким образом, действие этой команды состоит в перемещении на одну позицию вверх "B" нижних строк и в очистке нижней строки.

2.8.3. Временная задержка.

БЕЙСИК-оператор PAUSE можно легко эмулировать в машинном коде. Единицей измерения времени в "Спектруме" являются пятидесятые доли секунды (потому, что частота переменного тока в нашей сети - 50Гц. Если бы Вы жили в Англии, то у Вас в секунду проходило бы не 50, а 60 прерываний работы процессора, т.к. там частота сети равна 60 Гц).

Для того, чтобы исполнить паузу на m пятидесятых долей секунды, Вам достаточно загрузить это число m в регистровую пару BC, а затем вызвать процедуру ПЗУ PAUSE_1, расположенную по адресу 1F3DH (7997). Работая, процедура обеспечит заданную паузу. Во время своей работы процедура регулярно опрашивает системную переменную FLAGS (5C3BH = 23611). Ее пятый бит, если он включен, свидетельствует о том, что произошло нажатие клавиши. Таким образом, пауза может быть прервана нажатием любой клавиши - все, как в стандартном БЕЙСИКе.

Если Вы зашлете в регистровую пару BC ноль, то установленная пауза будет неограниченной, то есть до нажатия клавиши.

Вам, может быть, потребуется случай, когда пауза должна быть выдержана в течение заданного времени и не должна прерываться нажатием клавиши. Поможет несложная процедура:

```
78          PAUSE     LD A,B  ; Проверка BC на
B1                    OR C    ; ноль.
C8                    RET Z   ; Выход, если так.
0B                    DEC BC  ; Уменьшение BC.
76                    HALT    ; Задержка на 1/50
                              ; секунды (точнее - до
                              ; следующего прерыва-
                              ; ния, которое пройдет
                              ; чуть быстрее).
18F9                JR PAUSE  ; Возврат для повтора.
```

Зашлите в BC требуемую величину задержки и вызывайте эту процедуру.

2.8.4. Изображение точек.

Для изображения точек на экране из машинного кода есть два приема. Первый, и наиболее простой, состоит в том, чтобы загрузить координату X в регистр C, а координату Y - в регистр B, а затем вызвать процедуру PLOT_SUB, расположенную по адресу 22E5H (8933). Если у Вас координаты позиции печати точки уже содержатся в системной переменной COORDS (5C7D = 23677), то входить в эту процедуру можно в точке 22E8H (8936). Обратите внимание на то, что когда процедуры ПЗУ вычисляют по координатам позиции печати адрес соответствующего байта в дисплейной памяти, они коррумпируют регистр HL. Если он Вам нужен, его надо сохранить на стеке и потом восстановить.

Другой метод может быть более приемлем для тех энтузиастов, которые работают со встроенным калькулятором. Обе координаты должны быть предварительно помещены на вершину стека калькулятора в порядке: сначала x, затем y. Вызовом процедуры PLOT (22DCH = 8924) эти координаты снимаются со стека и точка печатается на экране.

2.8.5 Рисование линий.

Здесь также существуют два метода работы из машинного кода, но прежде, чем мы их рассмотрим, напомним, что в операторе БЕЙСИКа DRAW x,y параметры x и y не являются координатами экрана, а являются "смещением" точки конца отрезка относительно точки его начала. "Смещение" может быть не только положительным, но и отрицательным.

Метод 1.
~~~~~~~~

"Смещение" x загружается в регистры C и E. При этом в регистре C устанавливается его абсолютная величина - ABS (x), а в регистре E устанавливается 01, если "смещение" положительно или равно нулю или FFH, если отрицательно.

Аналогично "смещение" y загружается в регистры B и D. В регистре B - ABS (y), а в регистре D - 01 или FFH.

После этого можно рисовать линию, вызвав процедуру DRAW_LINE с точкой входа 24BAH (9402). Линия будет нарисована, но следует предусмотреть сохранение регистровой пары H'L' (альтернативная), т.к. она будет коррумпирована.

Метод 2.
~~~~~~~~

Второй путь, как и для печати точек, состоит в использовании калькулятора. Смещения x и y (в таком порядке) должны быть установлены на вершине стека калькулятора и тогда та же процедура DRAW_LINE, но с точкой входа 24B7 (9399) нарисует отрезок прямой на экране. H'L' здесь коррумпируется точно так же.

2.8.6 Рисование дуги.

Оператор БЕЙСИКа DRAW может вычерчивать между двумя точками не только прямые линии, но и соединять их с помощью дуги. Форма оператора - DRAW x,y,a. Параметр a определяет

угловую меру дуги, а x,y - "смещения". Угловая мера "a" задается в радианах.

Чтобы выполнить аналогичную задачу из машинного кода, следует все три параметра поместить на стек встроенного калькулятора в порядке x, y, a. Затем вызывается процедура DRAW_ARC (2394H = 9108). Дуга рисуется, как серия очень маленьких прямых, т.е. эта процедура в своей работе многократно вызывает процедуру DRAW_LINE и, следовательно, регистровая пара H'L' (альтернативная) также неизбежно коррумпируется.

2.8.7 Рисование окружности.

Здесь тоже ведется работа через встроенный калькулятор. Координаты центра окружности x и y, а затем радиус r должны быть помещены на вершину стека калькулятора, после чего выполняется вход в процедуру CIRCLE через точку входа 232DH (9005) для изображения окружности. H'L' - коррумпируется.

2.8.8 Сканирование экрана.

Проверка пиксела.

С помощью функции POINT (x,y) Вы можете проверить включен или выключен пиксел экрана с координатами x,y. Для имитации работы этой функции из машинного кода у Вас есть два пути. Во-первых, Вы можете прогрузить регистры B и C координатами x и y, после чего вызвать процедуру POINT_SUB с точкой входа (22CEH = 8910), а во-вторых, можете установить x и y на вершине стека калькулятора и вызвать эту же процедуру с точкой входа 22CBH (8907). И в том и в другом случае результат будет оставлен на вершине стека. Снять его оттуда можно, обратившись к процедуре ПЗУ FP_TO_A (2DD5H = 11733). Эта процедура перешлет результат в регистр A процессора. Если пиксел выключен, то есть имеет цвет PAPER, Вы получите 0, а если он включен, т.е. имеет цвет INK, Вы получите единицу.

Проверка атрибутов экрана.

С помощью функции ATTR (Y,X) Вы можете проверить установку атрибутов в заданном знакоместе с координатами Y,X. Здесь Y - номер экранного ряда (1...22), а X - номер столбца (1...32). Для имитации работы этой функции из машинного кода Вы можете загрузить регистры B и C координатами X и Y, после чего вызвать процедуру S_ATTR_S с точкой входа (2583H = 9603), а во-вторых, можете установить Y и X на вершине стека калькулятора и вызвать эту же процедуру с точкой входа 2580H (9600). И в том, и в другом случае результат будет оставлен на вершине стека. Снять его оттуда можно, обратившись к процедуре ПЗУ FP_TO_A (2DD5H = 11733), которая перешлет результат в аккумулятор. Анализируя данные по битам, Вы сможете установить параметры цветовых атрибутов в заданном знакоместе. Раскладку по битам см. на Рис.8 (стр.16).

Проверка содержимого заданного знакоместа.

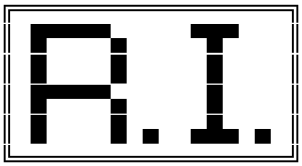
В БЕЙСИКе Вы можете воспользоваться функцией SCREEN$ (Y,X) для того, чтобы определить, есть ли какой-нибудь символ ASCII в данном знакоместе с координатами Y и X и если есть, то что это за символ. То есть, эта функция может исполнять сканирование экрана. Надо, правда, сказать, что работа этой функции в БЕЙСИ-Ке не отличается надежностью. В работе активно используется стек калькулятора, на котором может образовываться такая путаница, что например IF STRING$ (0,0)=STRING$(0,1) THEN PRINT "TRUE" будет всегда выдавать "TRUE" независимо от того, что содержится в знакоместах (0,0) и (0,1).

К счастью, мы можем организовать сканирование и из машинного кода, причем там такой путаницы не будет. Вы можете использовать процедуру ПЗУ S_SCRN$_S либо с точкой входа 2535H (9525), если Вы поместили Y и X на вершину стека калькулятора (именно в таком порядке), либо с точкой входа 2538H (9528), если Вы поместили в регистр C номер экранного ряда, а в B - номер столбца.

И в том, и в другом случае результат будет оставлен на вершине стека калькулятора. Чтобы переслать его оттуда в регистры микропроцессора, воспользуйтесь процедурой FP_TO_AEDCB (2BF1H = 11249). После этого в BC будет содержаться 0, если такого символа в наборе ASCII не существует или 1, если символ опознан. Узнать, что это за символ, можно, посмотрев содержимое аккумулятора - там будет содержаться его код.

Конечно, в результате сканирования смогут быть разысканы только символы ASCII - от 20H ("пробел") до 7FH ("копирайт"), т.к. только они имеются в таблице шрифта, на которую указывает системная переменная CHARS (23606 = 5C36H), но это дело можно доработать. Мы приведем процедуру, которая разыщет не только символы ASCII, но также и символы блочной графики и символы графики пользователя. Назовем ее SCANER и приведем чуть позже, в разделе 2.10. А пока надо оговорить еще несколько основополагающих моментов.

2.9. МЕТОДИКА РАСЧЕТА АДРЕСОВ В ЭКРАННОЙ ОБЛАСТИ ПО ЗАДАННЫМ КООРДИНАТАМ



Расчет адресов в экранной области памяти - это основная задача, встающая перед каждым, кто хочет заняться компьютерной графикой, работая в машинном коде. Не приобретя практики в этом вопросе трудно двигаться вперед.

2.9.1. Расчет адреса в дисплейном файле (координаты заданы в знакоместах).

Предположим, что координаты X и Y заданы в занкоместах и находятся в регистровой паре DE. В регистре D - координата X, а в регистре E - координата Y. Генеральной задачей является получить в регистровой паре HL адрес в дисплейном файле, соответствующий левому верхнему углу заданного знакоместа.

Приступая к этой задаче, напомним читателю как адрес в регистровой паре HL соответствует координатам экрана:

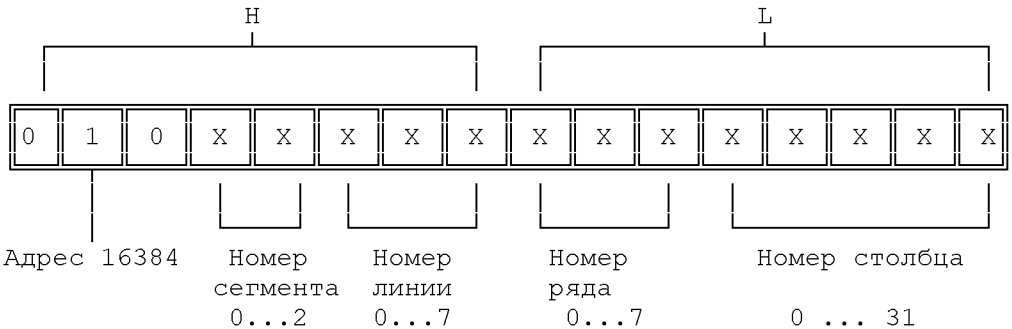
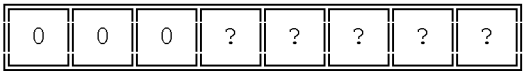


Рис. 16.

На первом этапе по координате Y определяется номер сегмента экрана. Т.к. координата Y не может быть больше 23-х, то для

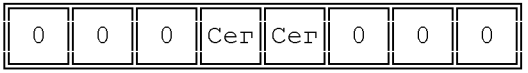
ее выражения достаточно 5-ти битов.

LD A,E



Поскольку в каждом сегменте по 8 рядов, то на номер сегмента указывают биты 3 и 4, поэтому замаскировав биты 0, 1, 2, получим указание на номер экранного сегмента:

AND 18



Командой OR 40 включим 6-ой бит, что послужит указанием на начало дисплейного файла:

OR 40

А теперь сравните полученную конструкцию с тем, что должно быть в регистре H (см. Рис.16). Таким образом, старший байт адреса мы уже сформировали.

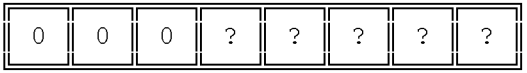
LD H,A

Выставили старший байт адреса в дисплейном файле. У нас, правда, занулены биты 0,1,2, указывающие на линию в ряду, но нас они и не интересуют, поскольку координаты заданы в знакоместах, а не в пикселах и верхнему левому углу знакоместа соответствует его нулевая (верхняя линия).

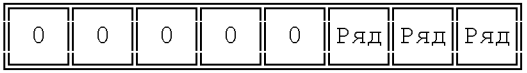
Теперь займемся вычислением младшего байта адреса для помещения его в регистр L.

LD A,E

Вновь рассматриваем координату Y.



AND 07

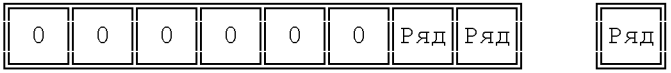


Оставив три младших бита, мы выделили тем самым номер ряда в экранном сегменте.

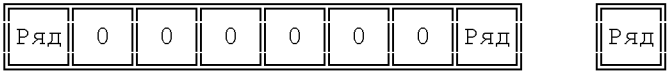
OR A

Обнулили флаг C в регистре F

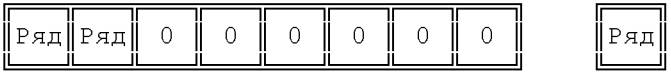
RRA



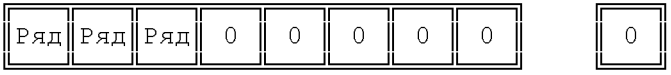
RRA



RRA



RRA



В результате четырех шагов вращения через флаг C три младших бита, указывавшие на номер ряда, стали старшими.

ADD A,D

Ряд	Ряд	Ряд	Ст	Ст	Ст	Ст	Ст

Прибавив значение столбца, а оно меньше 32 и потому занимает только пять младших битов, мы получили конструкцию, полностью соответствующую младшему байту адреса, изображенному на рис. 16 .

LD L,A

Выставили младший байт адреса и задача выполнена.

2.9.2. Расчет адреса в файле атрибутов.

Атрибуты "привязаны" только к знакоместам и потому координаты могут быть заданы только в знакоместах. Предположим, что координата Y содержится в регистре E, а координата X - в регистре D. Генеральная задача - сформировать в регистровой паре HL двухбайтный адрес, указывающий на ячейку памяти, в которой содерится байт атрибутов для нашего знакоместа.

Напомним, как адрес в регистровой паре HL соответствует атрибутам экрана.

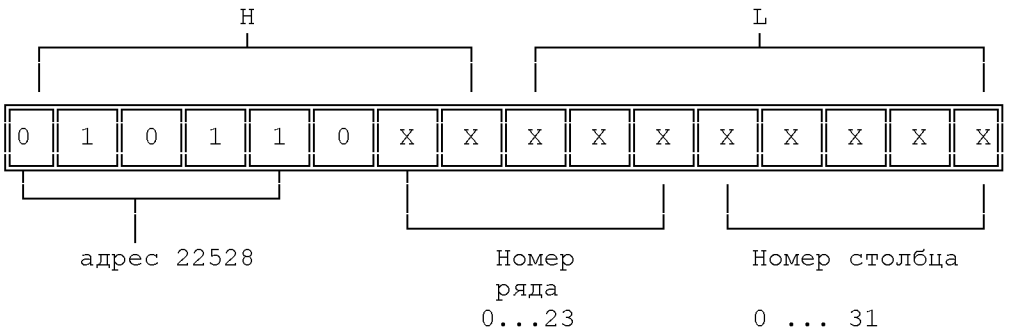


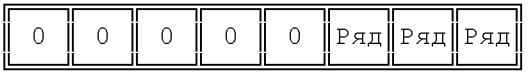
Рис. 17

Старший байт сформирован. Приступаем к формированию младшего байта. Вновь принимаем в аккумулятор координату Y.

LD A,E

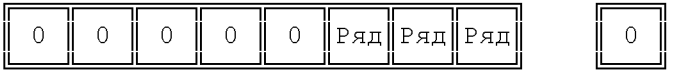
Оставляем три младших бита.

AND 07

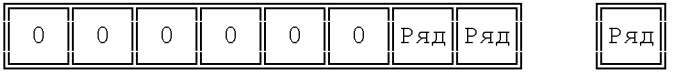


Обнуляем флаг C в регистре F. Флаг C.

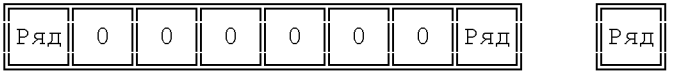
OR A



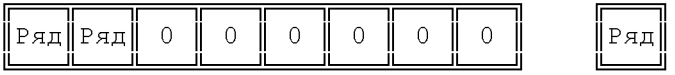
RRA



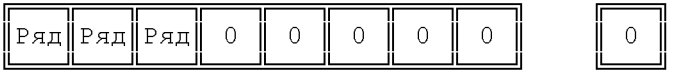
RRA



RRA



RRA



В результате четырех шагов вращения через флаг C три младших бита номера ряда стали старшими.

Принимаем в аккумулятор параметр Y.

LD A,E

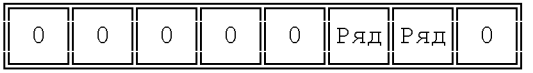
Оставляем от номера ряда два старших бита.

AND 18

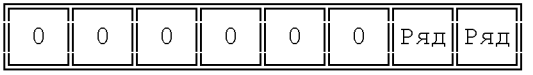
Тремя шагами логического вращения влево делаем эти два бита младшими.

SRL A

SRL A

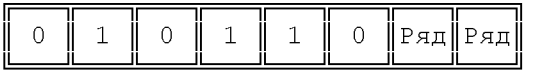


SRL A



Включаем биты 3,4,6 - они дают указание на начало файла атрибутов.

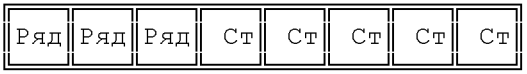
OR 58



Если сравнить полученную конструкцию с рис. 16, то видно, что она полностью соответствует старшему байту адреса, который должен быть в регистре H. Туда мы его и помещаем.

LD H,A

ADD A,D



Прибавив значение столбца (оно меньше 32 и потому занимает только пять младших битов), мы получили конструкцию, полностью соответствующую младшему байту адреса, изображенному на рис. 16 .

LD L,A

Выставили младший байт адреса - задача выполнена.

2.9.3. Расчет адреса в дисплейном файле (координаты заданы в пикселах).

Это наиболее трудоемкая задача из трех. Предположим, что координаты точки экрана содержатся в регистрах D (x) и E (y). Наша задача определить адрес в дисплейном файле, соответствующий данной точке. Надо сказать, что такая постановка не вполне корректна, т.к. минимальной единицей хранения данных является байт, а он определяет линию в знакоместе, т.е. 8 пикселов, а не один. Поэтому задача становится чуть глубже. Надо будет еще найти бит в байте, соответствующий заданной точке.

Попутно надо еще решить одну мелкую проблему. Дело в том, что координата y (когда она задана в пикселах) отсчитывается по экрану снизу вверх. Мы об этом уже говорили. А общая тенденция нарастания адресов в дисплейном файле - сверху вниз. Поэтому надо координату y преобразовать. Это делают путем подмены. Вместо y в проработку берут дополнение y до 175. Получается так, что y как бы инвертируется (назовем его y'). Тогда верхней линии экрана соответствует нулевое значение y', а нижней - максимальное значение.

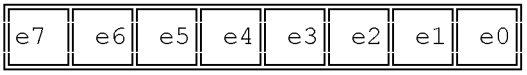
Загружаем в аккумулятор число 175:

LD A,0AFH

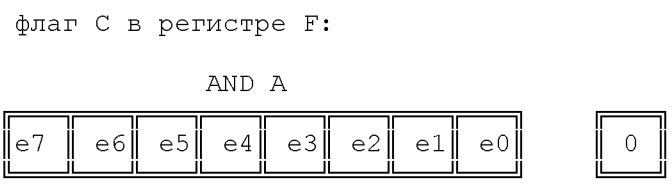
и вычитаем из него y:

```
SUB E
LD E,A
```

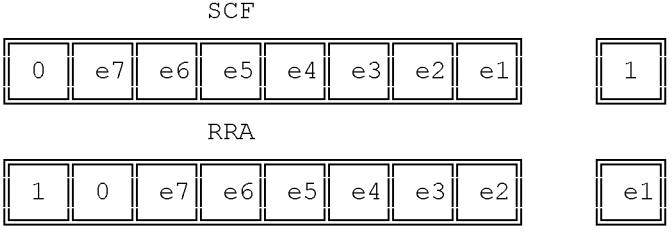
Теперь в регистре E у нас находится y'. Раскладка битов в регистре E и в аккумуляторе после этого может быть, например такой:



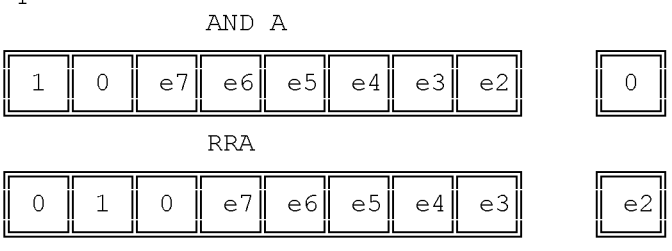
Выключим флаг C в регистре F:



Включим флаг C:



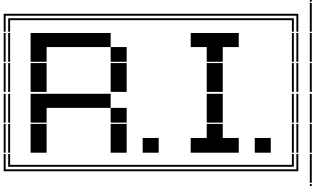
Выключим флаг C:



А теперь подменим биты 0, 1, 2 в аккумуляторе соответствующими им битами из регистра E:

```
XOR E
AND 0F8H
XOR E
```

ПРИМЕЧАНИЕ



Три последние операции XOR,AND,XOR образуют одну интересную комплексную операцию замещения. Если у Вас, например, есть два байта - A и B и Вы хотите несколько битов в байте A заменить на соответствующие биты из байта B, то эти операции позволят это сделать. Замещены будут те биты, которые замаскированы в операции AND нулями. Например пусть A имеет раскладку a7a6a5a4a3a2a1a0, а B имеет раскладку b7b6b5b4b3b2b1b0 и Вам надо заменить a5 и a3 на b5 и b3. Тогда в операции AND эти биты должны быть замаскированы.

```
Исходно: a7 a6 a5 a4 a3 a2 a1 a0

  XOR B: b7 b6 b5 b4 b3 b2 b1 b0
AND 214:  1  1  0  1  0  1  1  1
  XOR B: b7 b6 b5 b4 b3 b2 b1 b0
  ______________________________
Результат: a7 a6 b5 a4 b3 a2 a1 a0
```

В результате трех последних операций образуется следующая раскладка битов:

0	1	0	e7	e6	e2	e1	e0

Эта картина уже соответствует тому, что мы должны получить в регистре H (см. рис.15).

```
LD H,A
```

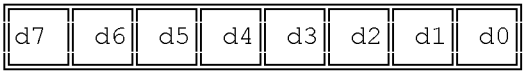
Таким образом, по дополнению координаты y до 175 можно определить номер сегмента экрана (e7,e6) и номер линии

(e2,e1,e0) в неизвестном (пока) ряду этого сегмента. Номер ряда мы установим, когда рассмотрим координату x и сможем заполнить регистр L.

Рассмотрим теперь координату x.

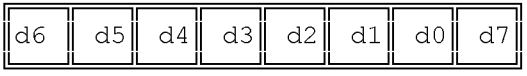
```
LD A,D
```

Раскладка битов в общем случае может быть например такой:

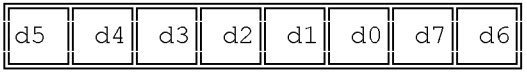


Делаем три шага вращения влево:

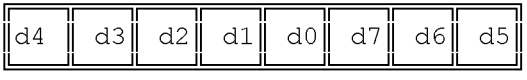
```
RLCA
```




```
RLCA
```



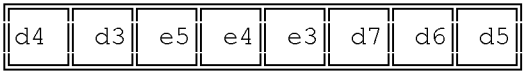

```
RLCA
```



И второй раз выполняем рассмотренную выше операцию логического замещения.

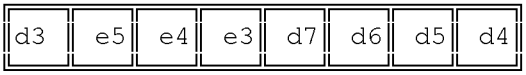
```
XOR E
AND C7
XOR E
```

В итоге получаем:

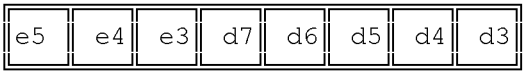


Еще два шага вращения влево:

RLCA



RLCA



Теперь конструкция в аккумуляторе полностью соответствует младшему байту адреса дисплейного файла (см. рис. 15).

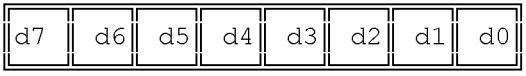
LD L,A

Младший байт адреса в дисплейном файле сформирован. В нем старшие три байта указывают на номер ряда внутри установленного сегмента экрана, а младшие пять байтов - на номер столбца.

Теперь мы знаем номер сегмента, ряда, столбца и линии. Все эти данные хранятся в регистровой паре HL. Последнее, что осталось сделать - это вспомнить, что данная линия в своем знакоместе имеет 8 пикселов, а нам нужен только один из них - необходимый. Фактически его номер определяется остатком от деления координаты x на 8, а вычисляется этот остаток - путем маскирования пяти старших битов, но есть еще один нюанс. Дело в том, что координата x изменяется слева направо 0,1,2... 255, а номера битов в байте, идут наоборот 7,6,5.....0. Поэтому нужно сделать преобразование.

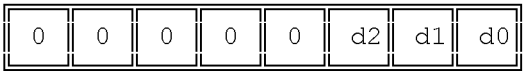
Возьмем координату x:

LD A,D



и замаскируем пять старших битов:

AND 07



Фактически мы имеем в аккумуляторе остаток от деления на 8. Поместим его в регистр B

```
LD B,A
```

и выполним преобразование:

```
LD A,08
SUB B
LD B,A
```

Итак, наша задача выполнена. В регистровой паре HL содержится адрес, по которому находится байт, задающий на экране линию, которой принадлежит наша исходная координата. А регистр B содержит номер бита, соответствующий нужной нам точке в этой линии.

2.10 УНИВЕРСАЛЬНАЯ ПРОГРАММА СКАНИРОВАНИЯ ЭКРАНА

Эта программа может выполнить сканирование экрана в заданном знакоместе и определить что за символ там содержится.

В отличие от функции БЕЙСИКа SCREEN$ (X,Y) и от ее машиннокодовых аналогов из системного ПЗУ компьютера, данная программа способна распознать и символы графики пользователя и символы блочной графики.

Программа имеет три различных точки входа. Какой из них пользоваться зависит от того, в каком виде заданы начальные условия. Если координаты сканируемого знакоместа заданы на вершине стека калькулятора в порядке Y,X, то процедура вызывается по метке SCR_FP. При этом результат подается на вершину стека калькулятора.

Если входные координаты задаются в качестве параметров функции пользователя FN S (), то точкой входа является метка SCR_FN. В этом случае сама функция пользователя должна быть задана как DEF FN S(X,Y) = USR addr, где addr - адрес метки SCR_FN. Вызов же процедуры тогда выполняется командой RANDOMIZE FN S(X,Y), где вместо X и Y Вы подставите конкретные параметры.

Если координаты задаются через регистры B (координата x) и C (координата y), то процедура вызывается по метке SCR_1. Это основная точка входа. Все остальные только выполняют преобразование входящих данных и обращаются сюда же.

Результат работы процедуры определяется состоянием флага Z (ZERO) регистра F. Если флаг включен, то сканирование прошло успешно и символ идентифицирован. Если флаг выключен - такого символа не существует. В случае успешного поиска код символа можно узнать в аккумуляторе процессора.

Программа SCANER.

```
SCR_FP  CALL 2307H          ;Вызов процедуры ПЗУ STK_TO_BC.
                            ;Процедура служит для размеще-
                            ;ния содержимого вершины стека
                            ;калькулятора в регистровой
                            ;паре BC.
        CALL SCR_1          ;Вызов другой точки входа. В
                            ;результате ее работы получим
                            ;в регистре A код символа, со-
                            ;держащегося в тестируемом зна-
                            ;коместе.
        LD BC,0000          ;Инициализация BC.
        JR NZ,SCR_STR       ;Если символ не идентифициро-
                            ;ван, то переход.
        INC BC              ;В паре BC теперь 1. Это длина
                            ;символьной строки, в которой
                            ;только один символ.
        RST 30              ;Вызов этой системной процедуры
                            ;ПЗУ резервирует пространство,
                            ;размер которого стоит в BC. На
                            ;адрес указывает DE.
        LD (DE),A           ;Заслали найденный символ.
SCR_STR JP 2AB2             ;Это точка входа системной
                            ;процедуры ПЗУ STACK_AEDCB,
                            ;которая помещает на вершину
                            ;стека калькулятора данные из
                            ;регистров A,E,D,C,B.
                            ;Эта процедура сама совершит и
                            ;возврат в вызывающую программу.

SCR_FN  LD HL,(DEFADD)      ;Указание на аргументы пользо-
                            ;вательской функции. Подроб-
                            ;ности см. в разделе 3.1.
        LD DE,0004
        ADD HL,DE           ;Сдвиг на 4 байта.
        LD C,(HL)           ;Координата Y.
        ADD HL,DE           ;
```

```
            ADD HL,DE           ;Сдвиг на 8 байтов.
            LD B,(HL)           ;Координата X.
            CALL SCR_1          ;Вызов другой точки входа. В
                                ;результате ее работы получим
                                ;в регистре A код символа, со-
                                ;держащегося в тестируемом зна-
                                ;коместе.
            JR Z,SCR_SINGL      ;Если символ найден - переход.
            XOR A               ;В противном случае в аккуму-
                                ;ляторе выставляем ноль.
SCR_SINGL   LD C,A              ;Подготовка к
            LD B,00             ;выходу.
            RET                 ;Выход.
```

Основной является точка входа SCR_1, т.к. остальные сводятся к ней же. Итак, BC содержит координаты X и Y.

```
SCR_1   LD A,C              ; ╓
        RRCA                ; ╟
        RRCA                ; ╟  Это поиск адреса зна-
        RRCA                ; ╟  коместа в дисплейной
        AND E0              ; ╟  памяти по заданным
        XOR B               ; ╟  координатам.
        LD E,A              ; ╟
        LD A,C              ; ╟  См. стр. 86...89
        AND 18              ; ╟
        XOR 40              ; ╟
        LD D,A              ; ╟
        PUSH DE             ; ╙
```

Итак, адрес знакоместа установлен в регистровой паре DE и сохранен на стеке. Теперь решается вопрос о том шаблон какого же символа изображен в этом знакоместе.

```
            LD HL, (CHARS)      ;Системная переменная CHARS
                                ;(23606 = 5C36) указывает на
                                ;256 байтов ниже, чем начало
                                ;таблицы шаблонов символов.
```

```
INC H               ;Теперь HL указывает точно
                    ;на начало таблицы шаблонов.
CALL 254DH          ;Адрес 254DH - одна из точек
                    ;входа в процедуру ПЗУ
                    ;S_SCRN_$, которая выполняет
                    ;проверку на то является ли
                    ;тестируемый шаблон символом
                    ;ASCII. Результат остается на
                    ;вершине стека калькулятора.
CALL 2BF1H          ;Адрес 2BF1 - точка входа про-
                    ;цедуры ПЗУ STK_FETCH, которая
                    ;снимает верхние пять значений
                    ;со стека калькулятора и раз-
                    ;носит их по регистрам
                    ;A,E,D,C,B.
                    ;Теперь в аккумуляторе имеется
                    ;код символа, если шаблон в тес-
                    ;тируемом знакоместе является
                    ;таковым.
POP DE              ;Восстановили адрес знакоместа.
DEC C
RET Z               ;Выход, если поиск был успешным.
```

Если имеющийся в данном знакоместе символ не является символом ASCII, начинаем проверять, а не является ли он символом графики пользователя UDG.

```
 PUSH DE            ;Запомнили адрес знакоместа.
 LD HL,(UDG)        ;Системная переменная UDG
                    ;(23675 = 5C7BH) указывает на
                    ;адрес, начиная с которого хра-
                    ;нятся шаблоны символов графики
                    ;пользователя.
 LD B,15H           ;15H = 21 DEC - количество сим-
                    ;волов графики пользователя.
 CALL 254F          ;Это точка входа в системную
                    ;процедуру S_SCRN_LP. Она про-
                    ;верит есть ли символ UDG, со-
```

```
                        ;ответствующий нашему знакомес-
                        ;ту и поместит результат на
                        ;вершину стека калькулятора.
        CALL 2BF1       ;Вход в STK_FETCH и разнесение
                        ;вершины стека калькулятора по
                        ;регистрам A,E,D,C,B.
```

Здесь надо сделать одно замечание. Дело в том, что на вершину стека, а затем и в акуумулятор пойдет не код символа UDG, а некоторое другое число, которое надо еще скорректировать. Причина в том, что процедура S_SCRN_LP в ПЗУ не предназначалась для работы с символами UDG, а только с символами ASCII, в результате чего возникает следующий эффект. Код найденного символа, помещаемый в аккумулятор, определяется как разность между числом 128 (предельный код символа ASCII) и тем, что осталось в счетчике проверяемых символов (регистр B). Так как символы UDG имеют конечный код равный 165, а не 128, то к полученному в аккумуляторе результату необходимо добавить недостающую разницу - число 37 (25H).

```
        ADD A,25        ;Выполнили коррекцию.
        POP HL          ;Теперь адрес знакоместа - в
                        ;паре HL.
        DEC C
        RET Z           ;Возврат, если поиск был ус-
                        ;пешным.
```

Если же поиск и на сей раз прошел безуспешно, нам остается только проверить: "А не является ли тестируемый символ символом блочной графики?" Проверка выполняется в два этапа. Сначала проверяется верхняя половина символа, а потом нижняя. Если первая проверка не прошла, то вторую выполнять нет смысла. Обратите также внимание на то, что у любого символа блочной графики 4 линии в каждой половине равны между собой и потому проверять можно не все четыре линии, а любую из них.

```
        CALL TEST_HALF  ;Проверка верхней половины
                        ;символа.
```

```
                RET NZ              ;Возврат, если проверка не
                                    ;прошла.
                LD C,A              ;Если проверка прошла, то в
                                    ;регистре C запоминается би-
                                    ;товая рскладка линии верхней
                                    ;половины символа.
                CALL TEST_HALF      ;Проверка нижней половины
                                    ;символа.
                RET NZ              ;Возврат, если проверка не
                                    ;прошла.
                ADD A,A             ;Здесь по конструкции символа
                ADD A,A             ;блочной графики определяется
                ADD A,C             ;его номер.
                ADD A,80            ;
                CP A                ;Включение флага Z как сигнал
                                    ;о том, что поиск был резуль-
                                    ;тативным.
                RET                 ;Окончательный выход в вызы-
                                    ;вающую программу.
```

Подпрограмма TEST_HALF выполняет тестирование верхней или нижней половины знакоместа в попытке определить не является ли помещенный там символ символом блочной графики.

```
 TEST_HALF      CALL TEST_LINE      ;Вызов подпрограммы, которая
                                    ;выполняет проверку первой
                                    ;линии данной половины.
                RET NZ              ;Возврат, если она не совпала.
                LD D,A              ;Если совпала, то ее код надо
                                    ;запомнить, ибо последующие
                                    ;три должны быть такими же.
                LD B,03             ;Счетчик цикла для трех линий.
    LOOP        CALL TEST_LINE      ;Вершина цикла для трех прове-
                                    ;рок линий.
                RET NZ              ;Возврат, если хотя бы одна из
                                    ;линий не соответствует блочной
                                    ;графике.
                CP D                ;Если она соответствует, но от-
```

```
                RET NZ              ;личается от первой, то тоже
                                    ;возврат.
                DJNZ LOOP           ;Конец цикла по линиям.
                RET                 ;Выход из подпрограммы.
```

Подпрограмма TEST_LINE проверяет соответствуют ли линии проверяемого знакоместа символам блочной графики. Конструкция линии задается одним байтом. Поскольку символы блочной графики имеют определенную симметрию, то многое можно о них сказать, проверив старший полубайт (4 бита) и младший полубайт (4 бита). Если один из них не соответствует конструкции символа блочной графики, то дальнейшее исследование символа бесполезно.

```
TEST_LINE   LD A,(HL)           ;HL указывает на адрес линии
                                ;в дисплейном файле. Теперь в
                                ;регистре A фактически содер-
                                ;жится конструкция линии.
            INC H               ;Переход к следующей линии.
            LD E,00             ;Инициализация регистра E.
            CALL TEST_NIBBLE    ;Проверка старшего полубайта.
            RET NZ              ;Возврат, если она не прошла.
            RLC E               ;Раскладка битов в регистре
                                ;E принимает вид:
                                ;  0 0 0 0 0 0 0 ah
                                ;где ah - состояние старшего
                                ;полубайта в тестируемой
                                ;линии.
            CALL TEST_NIBBLE    ;Проверка младшего полубайта.
            RET NZ              ;Возврат, если она не прошла.
            LD A,E              ;Аккумулятор имеет вид:
                                ;  al 0 0 0 0 0 0 ah, где
                                ;ah и al - состояние старшего
                                ;и младшего полубайтов в тес-
                                ;тируемой линии.
            RLCA                ;Аккумулятор имеет вид:
                                ;  0 0 0 0 0 0 ah al
                                ;и эта информация уже может
                                ;быть использована для
```

```
                                ;вычисления кода символа.
                CP A            ;Включение флага Z как сигнал
                                ;о том, что поиск был резуль-
                                ;тативным.
                RET             ;Возврат.
```

Подпрограмма TEST_NIBBLE проверяет полубайты каждой линии на предмет соответствия их конструкции символам блочной графики. При этом используется тот очевидный факт, что в любом символе блочной графики все четыре старших (или младших) бита должны быть одинаково включены или выключены.

```
TEST_NIBBLE     PUSH BC         ;Сохранили на стеке счетчик
                                ;линий в B, поскольку ниже
                                ;этот регистр будет использован
                                ;как счетчик битов в полубайте.
                RL E            ;В результате этой трехходовой
                RLA             ;манипуляции стрший бит E стал
                RR A            ;равен старшему биту A (копи-
                                ;рование произошло через флаг
                                ;переноса C.
                LD B,03         ;Организуем счетчик для провер-
                                ;ки трех последующих битов.
  LOOP_1        XOR E           ;Эта операция сбросит старший
                                ;бит аккумулятора, если он
                                ;равен (а сначала так оно и
                                ;есть) старшему биту в E.
                RLA             ;Очередной байт в аккумуляторе
                                ;становится старшим, а бывший
                                ;отправляется в флаг C
                JR C, EXIT      ;Флаг C мог включиться только
                                ;если один из трех битов, сле-
                                ;дующих за старшим, не равен
                                ;ему. Значит, это не символ
                                ;блочной графики и следует
                                ;переход на EXIT.
                DJNZ LOOP_1     ;Повторение для трех битов.
                CP A            ;Включение флага Z как сигнал
```

```
                                ;о том, что поиск был резуль-
                                ;тативным.
          JR EXIT_1             ;Подготовка к возврату.
EXIT      OR FF                 ;Выключение флага Z как сигнал
                                ;о том, что поиск был
                                ;безрезультатным.
EXIT_1    POP BC                ;Восстановление счетчика линий
                                ;в регистре B.
          RET                   ;Возврат.
```

3. ПРАКТИКУМ ПО ГРАФИКЕ В МАШИННЫХ КОДАХ

Эта обширная глава имеет триединую цель. Во-первых, она служит как бы практикумом по тому материалу, который был рассмотрен нами ранее. Во-вторых, это коллекция полезных идей, приемов и алгоритмов, являющихся учебным пособием для самоподготовки. И, наконец, в-третьих, тот материал, который здесь представлен, может быть легко инкорпорирован Вами в Ваши программы для достижения профессиональных графических эффектов.

Если Вы дочитали книгу до этого раздела, то уже хорошо себе представляете, что говоря о графике мы прибегаем к БЕЙСИКу только на уровне идей и концепций, а как только дело доходит до конкретных практических применений, нам приходится задействовать машинный код. И все-таки жаль расставаться с БЕЙСИКом, ведь с ним все так просто, что не использовать эту простоту было бы неразумно.

Желая сохранить простоту БЕЙСИКа и быстродействие машинно-кодовых процедур, мы организовали их совместное использование в рамках данной главы. БЕЙСИКу отводится роль связующего звена, а также блока логики верхнего уровня, а конкретные операции выполняет машинный код, представленный разными процедурами.

Существуют различные способы объединения в единой структуре и БЕЙСИКа и машинного кода. Так, на страницах наших книг и журналов мы не раз говорили о введении машинного кода в БЕЙСИК-строку за оператором REM. Вместе с тем, один из широко распространенных приемов, связанный с использованием функций пользователя FN до сих пор не нашел должного освещения в наших работах, о чем приходится сожалеть. Дело в том, что это тоже очень удобный прием, который становится особенно незаменимым в тех случаях, когда при вызове машиннокодовой процедуры из БЕЙСИКа надо передать в нее ряд параметров.

Обычный прием по передаче параметров из БЕЙСИКа в машинный код состоит в том, что в оперативной памяти Вы выделяете несколько ячеек памяти для хранения этих параметров, т.е. исполь-

зуете эти адреса, как адреса программных переменных. Затем, перед вызовом машиннокодовой процедуры, выставляете с помощью оператора POKE нужный Вам параметр в его ячейке памяти NN, вызываете процедуру, она "подхватывает" Ваш параметр из этой ячейки, отправляет в регистр процессора, работает с ним и, если нужно вернуть какой-то результат работы, тоже отправляет его в заранее зарезервированную ячейку памяти, например MM, после чего возвращается в БЕЙСИК. В БЕЙСИКе Вы можете с помощью оператора PEEK MM "прощупать", что же там наработала Ваша процедура.

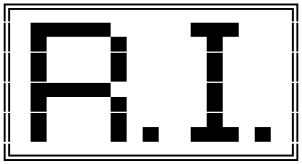
Это стандартный прием. Его используют, например, большие программы, целиком написанные в машинном коде. Так передаются параметры из одних процедур в другие. Для того, чтобы все это хорошо работало, программист обычно отводит солидный блок памяти (несколько сотен байтов) для хранения в нем таких программных переменных и строго следит, чтобы ничто этот блок во время работы программы не затерло.

Неудобство вполне очевидно - надо все время помнить, какой переменной какая ячейка памяти отведена, что откуда брать и куда класть результат, а также следить, чтобы временно использованные и по каким-то причинам измененные значения, своевременно восстанавливались и инициализировались. В принципе авторы выдающихся программ ведут целые досье (картотеки) на свои подпрограммы и на использованные для их работы переменные.

Если для программ, целиком написанных в машинном коде, это неудобство является необходимым злом, то для нашего случая, когда мы объединяем БЕЙСИК и машинный код в одной структуре, есть элегантный прием для передачи параметров, основанный на использовании в БЕЙСИКе функций, определенных пользователем и мы его сейчас рассмотрим, но сначала остановимся на вопросе о том, как хранятся в БЕЙСИКе функции пользователя.

ВНИМАНИЕ! Перед тем, как начинать работу с этой главой, необходимо внимательно изучить раздел 2.9.

3.1. Стандартный формат функции пользователя.



Давайте немного поэкспериментируем. Наберите на БЕЙСИКе следующую программу:

```
10 DEF FN a(x,y,z)=x+y+z
20 FOR i=23755 TO 24000
30 PRINT i, PEEK i
40 NEXT i
```

Вы, очевидно знаете, что начиная с адреса 23755 в памяти компьютера располагается текст БЕЙСИК-программы, поэтому вышеприведенная программа делает ничто иное, как распечатывает этот текст байт за байтом. Результат ее работы будет следующим:

```
23755         0  ┐
23756        10  ┘ - номер строки БЕЙСИКа;
23757        34  ┐
23758         0  ┘ - длина этой строки (34 байта);
23759       206     - код оператора DEF FN;
23760        97     - код буквы a;
23761        40     - код открывающей скобки;
23762       120     - код буквы x;
23763        14     - код, свидетельствующий, что следующие 5
                      байтов выражают некоторое действительное
                      число, записанное в интегральной форме;
23764       121  ┐
23765        44  │   полная "абракадабра", совершенно непо-
23766       122  ├─ - нятно, что это за число. То ли очень
23767        41  │   большое, то ли очень маленькое.
23768       120  ┘
23769        44     - код запятой;
23770       121     - код буквы y;
.................   и так далее.
```

Давайте поставим второй эксперимент:

```
10 DEF FN a(x,y,z)=x+y+z
15 LET test=FN a(3,8,5*2-7)
20 FOR i=23755 TO 24000
30 PRINT i, PEEK i
40 NEXT i
```

Мы добавили в программу строку 15, а теперь опять распечатаем текст строки 10, дав команду RUN. Вы получите результат:

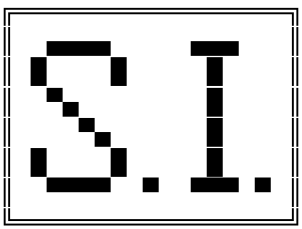
```
.........................................
23762       120     - код буквы x;
23763        14
23764         0 ─┐
23765         0  │
23766         3  ├─ -  код числа 3.
23767         0  │
23768         0 ─┘
23769        44     - код запятой;
23770       121     - код буквы y;
23771        14
23772         0 ─┐
23773         0  │
23774         8  ├─ -  код числа 8.
23775         0  │
23776         0 ─┘
.................. -  и т.далее
```

То, что мы сейчас увидели - довольно интересный результат. Фактически наличие в программе строки с номером 15 изменило содержание строки 10. С таким нечасто удается сталкиваться. А происходит это потому, что в "Спектруме", когда Вы задаете определение функции DEF FN и указываете ее параметры x,y,z,... и пр., за каждым именем параметра сразу резервируются 5 байтов, в которых впоследствии будут располагаться числовые значения этих параметров.

Как только встретился вызов функции FN в строке 15, эти параметры были вычислены и подставлены или просто подставлены, если их не надо вычислять, в забронированное место в строке 10.

Таким образом, непосредственно в БЕЙСИК-строке организован буфер, в котором хранятся текущие значения параметров. Сравните с программными переменными, текущие значения которых хранятся в специально отведенной области памяти, на которую указывает системная переменная VARS (23627 = 5C4B).

Так как ко времени вычисления функции пользователя параметры расставлены по своим местам, то их можно было бы использовать для передачи данных в машинный код, если бы удалось точно определить адреса этих мест или привязать их к чему-либо. И эта возможность имеется.



В наборе системных переменных компьютера есть системная переменная DEFADD (23563 = 5C0BH), которая в момент расчета функции пользователя содержит в себе адрес, с которого начинаются параметры этой функции. В прочие моменты времени там хранится 0. Если бы Вам удалось распечатать ее содержимое в момент расчета фунуции FN a,то Вы получили бы 23762, т.е. она указывает на имя первого параметра - "x". А дальше все просто. Если мы будем передавать через параметры процедуры целые числа, лежащие в интервале от 0 до 255, то первое число будет находиться по адресу (DEFADD)+4, там сейчас стоит число "3", второе - (DEFADD)+12 - там сечас стоит параметр y, равный восьми, третье - (DEFADD)+20 и т.д. с шагом по восемь байтов (один байт на букву, обозначающую имя этого параметра, один байт на запятую, разделяющую параметры, один байт на символ CHR14 и пять байтов на интегральную форму числа.)

Так можно использовать передачу параметров из БЕЙСИКа в машинный код через параметры функции пользователя. В нашем случае мы будем передавать целые числа от 0 до 255, т.е. один байт, но конечно можно передавать и любые действительные числа, что делает этот метод наиболее удобным. В этом случае переданная пятибайтная форма готова для обработки в калькуляторе.

3.2 Очистка заданного окна экрана.

Мы разберм эту процедуру самым подробнейшим образом. В отличие от обычной процедуры CLS, она позволяет очищать не весь экран, а заданное окно, параметры которого выставляются в БЕЙСИКе в функции FN a(x,y,h,v). В качестве начального адреса процедуры мы выбрали 63000, ее длина - 92 байта и еще четыре байта в конце процедуры отведены для хранения рабочих параметров. Список параметров:

x - горизонтальная координата левого верхнего угла окна, подлежащего очистке (задается в знакоместах 0...31);
y - вертикальная координата левого верхнего угла окна, подлежащего очистке (задается в знакоместах 0...23);
h - горизонтальный размер окна в знакоместах (h+x<32);
v - вертикальный размер окна в знакоместах (v+y<24).

Эта процедура может обслуживать весь экран, то есть все двадцать четыре символьных ряда, а не только главную его часть, т.е. 22 ряда. Если Вы зададите начальный параметр y больше, чем 23, процедура вернется в БЕЙСИК не работая, но на параметры x, h и v перехват ошибок не поставлен, чтобы не усложнять машинный код, так что здесь ошибка может привести к зависанию или сбросу компьютера, впрочем, при практическом использовании такой процедуры Вы сможете поставить перехват например в БЕЙСИКе перед вызовом FN a.

```
 10 REM *** Загрузчик машинного кода
 20 LET adr=63000: LET long=95: LET z=0
 30 FOR i=0 TO long-1: READ a
 40 POKE (adr+i),a: LET z=z+a
 50 NEXT i
 60 LET z=INT (((z/long)-INT (z/long))*long)
 70 READ a
 80 IF a<>z THEN PRINT "??": STOP
 90 REM
100 REM *** Пример использования процедуры
110 DEF FN a(x,y,h,v)=USR 63000
```

```
120 BORDER 4: PAPER 1: INK 6: CLS
130 FOR n=0 TO 703
140 PRINT "▀▄";
150 NEXT n
160 PAUSE 100
170 FOR n=1 TO 5
180 RANDOMIZE FN a(n*6-5,n*3-1,5,5)
190 NEXT n
200 REM *** Данные для машинного кода
210 DATA  42,  11,  92,   1,   4
220 DATA   0,   9,  86,   1,   8
230 DATA   0,   9   94, 237,  83
240 DATA 116, 246,   9,  86,   9
250 DATA  94, 237,  83, 118, 246
260 DATA 237,  91, 116, 246, 123
270 DATA 254,  23, 240, 237,  83
280 DATA 116, 246, 123, 230,  24
290 DATA 246,  64, 103, 123, 230
300 DATA   7, 183,  31,  31,  31
310 DATA  31, 130, 111,  58, 118
320 DATA 246,  71, 197, 229,   6
330 DATA   8, 197, 229,  58, 119
340 DATA 246,  71, 175, 119,  35
350 DATA  16, 252, 225, 193,  36
360 DATA  16, 240, 225, 193,  62
370 DATA  32, 133, 111,  48,   4
380 DATA  62,   8, 132, 103,  16
390 DATA 222, 201,   0,   0,   0
400 DATA  82
```

Дисассемблер машинного кода представлен ниже. Таким образом, фактически здесь и далее Вы имеете распечатку каждой процедуры, повторенную трижды. Первый раз в десятиричном виде в строках DATA, второй раз - то же самое в шестнадцатиричном коде и третий раз - в виде мнемоник АССЕМБЛЕРа. Такой тройной повтор сделан не только для того, чтобы каждый мог работать с тем кодом, который ему удобнее, но и из соображений надежности, в качестве дополнительной меры по борьбе с опечатками.

На первом этапе процедура принимает параметры x,y,h и v и перебрасывает их в ячейки 63092...63095.

```
63000   2A0B5C        LD HL,(5C0BH)   ;DEFADD - системная пере-
                                      ;менная, указывающая на
                                      ;то, где находятся пара-
                                      ;метры функции пользова-
                                      ;теля. Ее адрес 5C0BH
                                      ;(23563).
63003   010400        LD BC,0004      ;Сдвиг от DEFADD на 4 бай-
63006       09        ADD HL,BC       ;та (см. выше).
63007       56        LD D,(HL)       ;Координата x.
63008   010800        LD BC,0008      ;Сдвиг от DEFADD еще на 8
63012       09        ADD HL,BC       ;байтов (см. выше).
63013       5E        LD E,(HL)       ;Координата y.
63014 ED5374F6        LD(COORD),DE    ;Переброска параметров y
                                      ;и x в адреса 63092,63093.
63018       09        ADD HL,BC       ;Еще сдвиг на 8 байтов.
63019       56        LD D,HL         ;Ширина окна.
63020       09        ADD HL,BC       ;Еще сдвиг на 8 байтов.
63021       5E        LD E,(HL)       ;Высота окна.
63022 ED5376F6        LD(PARAM),DE    ;Переброска параметров v
                                      ;и h в адреса 63094,63095.
```

Теперь процедура проверяет не выходит ли параметр y за допустимый предел и начинает вычислять адрес в экранной области памяти, соответствующий координатам левого верхнего угла заданного окна.

```
63026 ED5B74F6        LD DE,(F674)    ;Координаты x и y.
63030       7B        LD A,E          ;Координата y.
63031     FE17        CP 17           ;Проверка на <=23.
63033       F0        RET P           ;Выход, если больше.
63034 ED5374F6        LD(F674),DE     ;Координаты x и y.
63038       7B        LD A,E          ;Координата y.
63039     E618        AND 18          ;╟ Расчет адреса
63041       F6        OR 40           ;╟ по координатам.
63042       67        LD H,A          ;╟
```

```
63043       7B        LD A,E          ;╟
63044     E607        AND 07          ;╟
63046       B7        OR A            ;╟
63047       1F        RRA             ;╟
63048       1F        RRA             ;╟
63049       1F        RRA             ;╟
63050       1F        RRA             ;╟
63051       82        ADD A,D         ;╟
63052       6F        LD L,A          ;╟
```

Теперь регистры H и L полностью соответствуют рис. 16 на стр. 86, т.е. мы выставили в HL адрес дисплейной памяти, соответствующий нулевой линии левого верхнего угла окна. Наша следующая задача - непосредственная очистка окна, которую мы будем делать, засылая нули в ячейки памяти, соответствующие линиям входящих в окно знакомест.

Здесь нам придется организовать несколько вложенных циклов. Первый цикл (внешний) - по вертикали, то есть по рядам от 1 до параметра v. Второй цикл (средний) - по линиям в ряду - от 1 до 8. И третий цикл (внутренний) - по горизонтали, то есть по столбцам - от 1 до параметра h.

```
63053   3A76F6        LD A,(F676)     ;Параметр v
63056       47        LD B,A          ;В регистре "B" он будет
                                      ;счетчиком цикла.
63057       C5 LOOP_V PUSH BC         ;Сохранили его на стеке
63058       E5        PUSH HL         ;Сохранили текущий адрес.
63059     0608        LD B,08         ;Счетчик цикла по линиям.
63061       C5 LOOP_8 PUSH BC         ;Сохранили его на стеке
63062       E5        PUSH HL         ;Сохранили на стеке
                                      ;текущий адрес.
63063   3A77F6        LD A,(F677)     ;Параметр h
63066       47        LD B,A          ;Счетчик цикла по столбцам
63067       AF        XOR A           ;Это простейший способ об-
                                      ;нуления аккумулятора.
63068       77 LOOP_H LD(HL),A        ;Начало цикла по столбцам.
                                      ;Очистка одной линии.
```

```
63069       23          INC HL          ;Переход на соседнее зна-
                                        ;коместо враво.
63070     10FC          DJNZ LOOP_H     ;Конец цикла по столбцам.
63072       E1          POP HL          ;Восстановление данных со
63073       C1          POP BC          ;стека.
63074       24          INC H           ;Переход к следующей линии
                                        ;в данном ряду.
63075     10F0          DJNZ LOOP_8     ;Конец цикла по линиям.
63077       E1          POP HL          ;Восстановление данных со
63078       C1          POP BC          ;стека.
63079     3E20          LD A,20         ;Переход
63081       85          ADD A,L         ;на следующий
63082       6F          LD L,A          ;ряд.
63083     3004          JR NC,NO_SEG    ;Здесь может возникнуть
                                        ;переход из сегмента в се-
                                        ;гмент. Если этого нет, то
                                        ;переход на метку NO_SEG.
63085     3E08          LD A,08         ;Корректировка
63087       84          ADD A,H         ;в HL
63088       67          LD H,A          ;номера сегмента.
63089     10DE NO_SEG   DJNZ LOOP_V     ;Конец цикла по рядам.
63091       C9          RET             ;Выход из программы.
63092     0000 COORD    DEFW 0000
63094     0000 PARAM    DEFW 0000
```

3.3. Окрашивание заданного окна цветом INK.

Аналогично предыдущей процедуре, параметры заданного окна, выставляются в БЕЙСИКе с помощью пользовательской функции FN b(x,y,h,v,c,b,f). Начальным адресом взят для примера адрес 62800, длина процедуры - 127 байтов и еще пять байтов в конце отведены для хранения рабочих переменных. Список параметров:

x - горизонтальная координата левого верхнего угла окна, подлежащего очистке (задается в знакоместах 0...31);
y - вертикальная координата левого верхнего угла окна, подлежащего очистке (задается в знакоместах 0...23);

h - горизонтальный размер окна в знакоместах (h+x<32);
v - вертикальный размер окна в знакоместах (v+y<24).
c - заданный цвет INK (0...7);
b - признак яркости BRIGHT (0,1);
f - признак мигания FLASH (0,1).

Эта процедура также может обслуживать весь экран - все 24 строки, а не только 22 строки основной его части.

```
 10 REM *** Загрузчик машинного кода
 20 LET adr=62800: LET long=130: LET z=0
 30 FOR i=0 TO long-1: READ a
 40 POKE (adr+i),a: LET z=z+a
 50 NEXT i
 60 LET z=INT (((z/long)-INT (z/long))*long)
 70 READ a
 80 IF a<>z THEN PRINT "??": STOP
 90 REM
100 REM *** Пример использования процедуры
110 DEF FN b(x,y,h,v,c,b,f)=USR 62800
120 BORDER 0: PAPER 0: INK 4: CLS
130 FOR n=0 TO 703
140 PRINT "▚";
150 NEXT n
160 PAUSE 100
170 FOR n=1 TO 7
180 RANDOMIZE FN b(0,n*3-3,32,3,8-n,0,0)
190 NEXT n
200 FOR n=1 TO 7
210 RANDOMIZE FN b(n*4-1,0,2,22,n,0,0)
220 NEXT n
300 REM *** Данные для машинного кода
310 DATA  42,  11,  92,   1,   4
320 DATA   0,   9,  86,   1,   8
330 DATA   0,   9   94, 237,  83
340 DATA 210, 245,   9,  86,   9
350 DATA  94, 237,  83, 208, 245
360 DATA   9, 126, 230,   7,  50
```

```
370 DATA 207, 245,   9, 126, 230
380 DATA   1,  40,   8,  58, 207
390 DATA 245, 246,  64,  50, 207
400 DATA 245,   9, 126, 230,   1

410 DATA  40,   8,  58, 207, 245
420 DATA 246, 128,  50, 207, 245
430 DATA 237,  91, 210, 245,  58
440 DATA 208, 245, 254,   0, 200
450 DATA 237,  83, 210, 245, 123
460 DATA 230,  24, 203,  63, 203
470 DATA  63, 203,  63, 246,  88
480 DATA 103, 123, 230,   7, 183
490 DATA  31,  31,  31,  31, 130
500 DATA 111,  58, 208, 245,  71

510 DATA 197, 229,  58, 209, 245
520 DATA  71, 126, 230,  56,  79
530 DATA  58, 207, 245, 177, 119
540 DATA  35,  16, 244, 225,   1
550 DATA  32,   0,   9, 193,  16
560 DATA 230, 201,   0,   0,   0
570 DATA  46,   0
```

Основным отличием этой процедуры от предыдущей является то, что она демонстрирует не операции с памятью дисплейного файла, а операции с файлом атрибутов. Теперь начальный адрес в HL выставляется в соответствии с файлом атрибутов (см. рис.17 на стр. 89).

Здесь для выполнения работы достаточно не трех циклов, как при обслуживании дисплейного файла, а только двух - внешнего цикла по вертикали (параметр v) и внутреннего - по горизонтали (параметр h), т.к. цикл по восьми линиям уже не нужен.

Обратите внимание на то, что под три параметра c,b и f отводится всего одна ячейка памяти. В ней параметру c отведены биты 0,1,2, параметру b - бит 6 и параметру f - бит 7. Все в

полном соответствии с раскладкой атрибутов по битам (рис. 8).

Дисассемблер программы:

```
62800   2A0B5C        LD HL,(5C0BH)  ;См. с. 109...111?
62803   010400        LD BC,0004     ;Сдвиг от DEFADD на 4 бай-
62806       09        ADD HL,BC      ;та (см. с. 109...111).
62807       56        LD D,(HL)      ;Координата x.
62808   010800        LD BC,0008     ;Сдвиг от DEFADD еще на 8
62811       09        ADD HL,BC      ;байтов (см.с.109...111)
62812       5E        LD E,(HL)      ;Координата y.
62813 ED53D2F5        LD(COORD),DE   ;Переброска параметров y
                                     ;и x в адреса 62930,62931.
62817       09        ADD HL,BC      ;Еще сдвиг на 8 байтов.
62818       56        LD D,HL        ;Ширина окна.
62819       09        ADD HL,BC      ;Еще сдвиг на 8 байтов.
62820       5E        LD E,(HL)      ;Высота окна.
62821 ED53D0F5        LD(PARAM),DE   ;Переброска параметров v
                                     ;и h в адреса 62928,62929.
62825       09        ADD HL,BC      ;Следующий параметр (INK)
62826       7E        LD A,(HL)      ;помещен в аккумулятор.
62827     E607        AND 07         ;и переправлен в отведен-
62829   32CFF5        LD (INK),A     ;ную ему ячейку 62927.
62832       09        ADD HL,BC      ;Параметр (BRIGHT)
62833       7E        LD A,(HL)      ;принимается,
62834     E601        AND 01         ;выделяется,
62836     2808        JR Z,SKIP_1    ;и, если не равен нулю,
62838   3ACFF5        LD A,(INK)     ;то в параметре INK
62841     F640        OR 40          ;включается 6-ой бит
62843   32CFF5        LD (INK),A     ;и INK сохраняется.
62846       09 SKIP_1 ADD HL,BC      ;Параметр (FLASH)
62847       7E        LD A,(HL)      ;принимается,
62848     E601        AND 01         ;выделяется,
62850     2808        JR Z,SKIP_2    ;и, если не равен нулю,
62852   3ACFF5        LD A,(INK)     ;то в параметре INK
62855     F680        OR 80          ;включается 7-ой бит
62857   32CFF5        LD (INK),A     ;и INK сохраняется.
62860 ED5BD2F5 SKIP_2 LD DE,(COORD)  ;Координаты x,y.
```

```
62864   3AD0F5          LD A,(F5D0)     ;Высота окна (v)
62867     FE00          CP 00           ;Проверка высоты на ноль.
62869       C8          RET Z           ;Выход, если так
62870 ED53D2F5          LD (COORD),DE   ;Координаты y,x.
62874       7B          LD A,E          ||
62875     E618          AND 18          ||
62877     CB3F          SRL A           ||  Расчет адреса по
62879     CB3F          SRL A           ||  координатам.
62881     CB3F          SRL A           ||
62883     F658          OR 58           ||
62885       67          LD H,A          ||
62886       7B          LD A,E          ||
62887     E607          AND 07          ||
62889       B7          OR A            ||
62890       1F          RRA             ||
62891       1F          RRA             ||
62892       1F          RRA             ||
62893       1F          RRA             ||
62894       82          ADD A,D         ||
62895       6F          LD L,A          ||
62896   3AD0F5          LD A,(F5D0)     ;Параметр v
62899       47          LD B,A          ;становится параметром
62900       C5 LOOP_V   PUSH BC         ;цикла и сохраняется на
62901       E5          PUSH HL         ;стеке вместе с адресом.
62902   3AD1F5          LD A,(F5D1)     ;Параметр h становится
62905       47          LD B,A          ;параметром цикла.
62906       7E LOOP_H   LD A,(HL)       ;Сняли с экрана атрибуты
                                        ;знакоместа.
62907     E638          AND 38          ;Оставили включенными
                                        ;только биты, отвечающие
                                        ;за цвет PAPER.
62909       4F          LD C,A          ;Временно запомнили в "C".
62910   3ACFF5          LD A,(INK)      ;Параметры c,b,f.
62913       B1          OR C            ;Наложили их на PAPER.
62914       77          LD (HL),A       ;Изменили атрибуты.
62915       23          INC HL          ;Следующее знакоместо.
62916     10F4          DJNZ LOOP_H     ;Конец цикла по горизон-
                                        ;тали.
```

```
62918       E1          POP HL          ;Адрес начала ряда.
62919   012000          LD BC,0020      ;Переход на
62922       09          ADD HL,BC       ;следующий ряд.
62923       C1          POP BC          ;Параметр цикла по v.
62924     10E6          DJNZ LOOP_V     ;Конец цикла по v.
62926       C9          RET             ;Выход.
62927       00 INK      DEFB 00         ;Атрибуты
62928     0000 PARAM    DEFW 0000       ;Параметры v и h.
62930     0000 COORD    DEFW 0000       ;Параметры y и x.
```

3.4. Окрашивание заданного окна цветом PAPER.

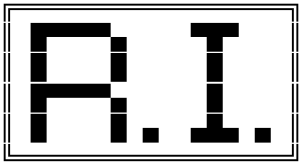
Процедура работает совершенно аналогично предыдущей, за исключением того, что вместо цвета INK, в окне устанавливается заданный цвет PAPER. Параметры окна выставляются в БЕЙСИКе с помощью пользовательской функции FN c(x,y,h,v,c,b,f). Начальным адресом взят для примера адрес 62600, длина процедуры - 139 байтов и еще пять байтов в конце отведены для хранения рабочих переменных. Список параметров тот же, что и в предыдущей процедуре (см.с. 116), единственное отличие состоит в том, что параметр "c" содержит информацию о цвете PAPER (от 0 до 7), а не INK, как было ранее.

Эта процедура также может обслуживать весь экран - все 24 строки, а не только 22 строки основной его части.

```
 10 REM *** Загрузчик машинного кода
 20 LET adr=62600: LET long=145: LET z=0
 30 FOR i=0 TO long-1: READ a
 40 POKE (adr+i),a: LET z=z+a: NEXT i
 60 LET z=INT (((z/long)-INT (z/long))*long)
 70 READ a
 80 IF a<>z THEN PRINT "??": STOP
 90 REM *** Пример использования процедуры
100 DEF FN c(x,y,h,v,c,b,f)=USR 62600
110 BORDER 1: PAPER 4: CLS
120 FOR i=1 TO 120
130 LET x1=INT (RND*17)
```

```
140 LET y1=INT (RND*10)
150 LET h1=INT (RND*16)
160 LET v1=INT (RND*15)
170 LET c1=INT (RND*7)
180 RESTORE FN c(x1,y1,h1,v1,c1,0,0)
190 NEXT i: PAUSE 0 : REM пауза до нажатия клавиши.
200 REM *** Данные для машинного кода
210 DATA  42,  11,  92,   1,   4
220 DATA   0,   9,  86,   1,   8
230 DATA   0,   9   94, 237,  83
240 DATA  22, 245,   9,  86,   9
250 DATA  94, 237,  83,  20, 245
260 DATA   9, 126, 230,   7, 203
270 DATA  39, 203,  39, 203,  39
280 DATA  50,  19, 245,   9, 126
290 DATA 230,   1,  40,   8,  58
300 DATA  19, 245, 246,  64,  50
310 DATA  19, 245,   9, 126, 230
320 DATA   1,  40,   8,  58,  19
330 DATA 245, 246, 128,  50,  19
340 DATA 245, 237,  91,  22, 245
350 DATA  58,  20, 245, 254,   0
360 DATA 200,  58,  21, 245, 254
370 DATA   0, 200, 237,  83,  22
380 DATA 245, 123, 230,  24, 203
390 DATA  63, 203,  63, 203,  63
400 DATA 246,  88, 103, 123, 230
410 DATA   7, 183,  31,  31,  31
420 DATA  31, 130, 111,  58,  20
430 DATA 245,  71, 197, 229,  58
440 DATA  21, 245,  71, 126, 230
450 DATA   7,  79,  58,  19, 245
460 DATA 177, 119,  35,  16, 244
470 DATA 225,   1,  32,   0,   9
480 DATA 193,  16, 230, 201,   0
490 DATA   0,   0,   0,   0,   0
500 DATA  12
```

Демонстрационная программа выполняет печать случайных цветных окон, окрашенных цветом PAPER. Мы бы хотели обратить Ваше особое внимание на то, что процедура вызывается не через RANDOMIZE USR, а через RESTORE USR. В чем здесь разница? С точки зрения самой процедуры, ей это все равно, для нее разницы нет, но при использовании RANDOMIZE USR возникает косвенный эффект,



который состоит в том, что всякий раз, когда компьютер встречает оператор RANDOMIZE, он переустанавливает содержимое системной переменной SEED (23670=5C76H), содержащей базовую величину для генерации случайных чисел. Если в большинстве случаев нам это безразлично, то здесь мы используем случайные числа для того, чтобы задавать параметры окрашиваемого окна (строки 130.. .170). Если бы на каждом шаге функция RANDOMIZE одинаково переинициализировала бы нам системную переменную SEED, то мы ничего нового на экране не увидели бы. Таким образом, если Вы имеете дело со случайными числами, вычисляемыми через RND, то использовать RANDOMIZE USR нельзя - пользуйтесь по обстоятельствам RESTORE USR, PRINT USR и т.п.

Замечательный эффект в духе Малевича и Мондриана Вы сможете получить, например, если используете эту процедуру со следующей БЕЙСИк-программой. Подобные приемы придадут Вашим программам неповторимый колорит.

```
100 DEF FN c(x,y,h,v,c,b,f)=USR 62600: BORDER 7: PAPER 7: CLS
110 RANDOMIZE FN c(4,10,1,14,0,0,0)
120 RANDOMIZE FN c(16,0,1,24,0,0,0)
130 RANDOMIZE FN c(24,0,1,24,0,0,0)
140 RANDOMIZE FN c(0,9,32,1,0,0,0)
150 RANDOMIZE FN c(0,17,32,1,0,0,0)
160 RANDOMIZE FN c(14,3,18,1,0,0,0)
170 RANDOMIZE FN c(0,0,16,9,4,1,0)
180 RANDOMIZE FN c(5,18,11,6,6,1,0)
190 RANDOMIZE FN c(25,0,7,9,2,1,0)
195 RANDOMIZE FN c(17,10,7,7,1,1,0): PAUSE 0
```

Дисассемблер программы:

```
62600   2A0B5C        LD HL,(5C0BH)  ;См. с.109...111.
62603   010400        LD BC,0004     ;Сдвиг от DEFADD на 4 бай-
62606       09        ADD HL,BC      ;та (см. с.109...111 ).
62607       56        LD D,(HL)      ;Координата x.
62608   010800        LD BC,0008     ;Сдвиг от DEFADD еще на 8
62611       09        ADD HL,BC      ;байтов (см.с.109...111).
62612       5E        LD E,(HL)      ;Координата y.
62613 ED5316F5        LD(COORD),DE   ;Переброска параметров y
                                     ;и x в адреса 62742,62743.
62617       09        ADD HL,BC      ;Еще сдвиг на 8 байтов.
62618       56        LD D,HL        ;Ширина окна.
62619       09        ADD HL,BC      ;Еще сдвиг на 8 байтов.
62620       5E        LD E,(HL)      ;Высота окна.
62621 ED5314F5        LD(PARAM),DE   ;Переброска параметров v
                                     ;и h в адреса 62740,62741.
62625       09        ADD HL,BC      ;Параметр PAPER помещен в
62626       7E        LD A,(HL)      ;аккумулятор      00000???
62627     E607        AND 07         ;Выделение PAPER. 00000???
62629     CB27        SLA A          ;Сдвиг влево.     0000???0
62631     CB27        SLA A          ;Сдвиг влево.     000???00
62633     CB27        SLA A          ;Сдвиг влево.     00???000
62635   3213F5        LD (PAPER),A   ;
62638       09        ADD HL,BC      ;Параметр (BRIGHT)
62639       7E        LD A,(HL)      ;принимается,
62640     E601        AND 01         ;выделяется,
62642     2808        JR Z,SKIP_1    ;и, если не равен нулю,
62644   3A13F5        LD A,(PAPER)   ;то в параметре PAPER
62647     F640        OR 40          ;включается 6-ой бит
62649   3213F5        LD (PAPER),A   ;и PAPER сохраняется.
62652       09 SKIP_1 ADD HL,BC      ;Параметр (FLASH)
62653       7E        LD A,(HL)      ;принимается,
62654     E601        AND 01         ;выделяется,
62656     2808        JR Z,SKIP_2    ;и, если не равен нулю,
62658   3A13F5        LD A,(PAPER)   ;то в параметре PAPER
62661     F680        OR 80          ;включается 7-ой бит
62663   3213F5        LD (PAPER),A   ;и PAPER сохраняется.
```

```
62666 ED5B16F5 SKIP_2 LD DE,(COORD)  ;Координаты x,y.
62670   3A14F5        LD A,(F514)    ;Высота окна (v).
62673     FE00        CP 00          ;Проверка высоты на ноль.
62675       C8        RET Z          ;Выход, если так
62676   3A15F5        LD A,(F515)    ;Ширина окна (h).
62679     FE00        CP OO          ;Проверка ширины на ноль.
62681       C8        RET Z          ;Выход, если так
62682 ED5322F5        LD (COORD),DE  ;Координаты y,x.
62686       7B        LD A,E         ‖
62687     E618        AND 18         ‖ Определение адреса
62689     CB3F        SRL A          ‖ по координатам.
62691     CB3F        SRL A          ‖
62693     CB3F        SRL A          ‖
62695     F658        OR 58          ‖
62697       67        LD H,A         ‖
62698       7B        LD A,E         ‖
62699     E607        AND 07         ‖
62701       B7        OR A           ‖
62702       1F        RRA            ‖
62703       1F        RRA            ‖
62704       1F        RRA            ‖
62705       1F        RRA            ‖
62706       82        ADD A,D        ‖
62707       6F        LD L,A         ‖
62708   3A14F5        LD A,(F514)    ;Параметр v
62711       47        LD B,A         ;становится параметром
62712       C5 LOOP_V PUSH BC        ;цикла и сохраняется на
62713       E5        PUSH HL        ;стеке вместе с адресом.
62714   3A15F5        LD A,(F515)    ;Параметр h становится
62717       47        LD B,A         ;параметром цикла.
62718       7E LOOP_H LD A,(HL)      ;Сняли с экрана атрибуты
                                     ;знакоместа.
62719     E607        AND 07         ;Оставили включенными
                                     ;только биты, отвечающие
                                     ;за цвет INK.
62721       4F        LD C,A         ;Временно запомнили в "C".
62722   3A13F5        LD A,(PAPER)   ;Параметры c,b,f.
62725       B1        OR C           ;Наложили их на INK.
```

```
62726       77          LD (HL),A       ;Изменили атрибуты в
                                        ;знакоместе.
62727       23          INC HL          ;Следующее знакоместо.
62728     10F4          DJNZ LOOP_H     ;Конец цикла по горизон-
                                        ;тали.
62730       E1          POP HL          ;Адрес начала ряда.
62731   012000          LD BC,0020      ;Переход на
62734       09          ADD HL,BC       ;следующий ряд.
62735       C1          POP BC          ;Параметр цикла по v.
62736     10E6          DJNZ LOOP_V     ;Конец цикла по v.
62738       C9          RET             ;Выход.
62739       00 PAPER    DEFB 00         ;Атрибуты PAPER, BRIGHT,
                                        ;FLASH.
62740     0000 PARAM    DEFW 0000       ;Параметры v и h.
62742     0000 COORD    DEFW 0000       ;Параметры y и x.
```

3.5. Масштабное увеличение текста. Печать по горизонтали.

Удвоение размеров символов на "Спектруме" имеет в своей основе очень простые принципы. Представим, что символ помещен на сетку 8X8 пикселов. Тогда при переносе его на сетку 16X16 программа должна найти включенный пиксел и вместо него напечатать два пиксела по горизонтали и два по вертикали.

Приведенная ниже программа использует этот подход для того, чтобы изобразить символами двойного размера заданную символьную строку. Процедура FN d(x,y) выполняет печать текста по горизонтали символами двойной ширины и двойной высоты. Здесь x и y задают начальное знакоместо для печати текста.

Процедура FN d(x,y) имеет длину 220 байтов. В конце этой процедуры отводятся еще несколько байтов для размещения в них программных переменных. Мы выбрали для этой процедуры начальным адресом 62200, а память, начиная с адреса 62500 по 62599 выделили в качестве буфера, в котором хранится текст сообщения, подлежащего печати. Текст хранится здесь в виде последователь-

ности символов и отправляется туда из БЕЙСИКа при посредстве символьной переменной n$.

```
 10 REM *** Загрузчик машинного кода
 20 LET adr=62200: LET long=220: LET z=0
 30 FOR i=0 TO long-1: READ a
 40 POKE (adr+i),a: LET z=z+a
 50 NEXT i
 60 LET z=INT (((z/long)-INT (z/long))*long)
 70 READ a
 80 IF a<>z THEN PRINT "??": STOP
 90 REM *** Пример использования процедуры
100 DEF FN d(x,y)=USR 62200
110 BORDER 1: PAPER 6: INK 1: CLS
120 FOR i=0 TO 13
130 DRAW 255,0: DRAW 0,6
140 DRAW -255,0: DRAW 0,6
150 NEXT i
160 DRAW 255,0
170 LET n$="SPECTRUM"
180 GO SUB 230
190 FOR m=1 TO 22 STEP 4
200 RANDOMIZE FN d(x,y)
210 NEXT m
220 PAUSE 0
230 LET t=LEN(n$)
240 LET k=62499
250 FOR i=1 TO t
260 LET n=CODE n$(i)
270 POKE k+i,n
280 NEXT i
290 POKE k+i,13
299 RETURN

300 REM *** Данные для машинного кода
310 DATA  42,  11,  92,   1,   4
320 DATA   0,   9,  86,   1,   8
330 DATA   0,   9,  94, 237,  83
```

```
340 DATA 240, 243,  62,  99,  71
350 DATA  33,  36, 244,  34, 244
360 DATA 243, 197, 237,  91, 240
370 DATA 243,  62,  30, 186, 242
380 DATA  37, 243,  22,   0,  28
390 DATA  28, 237,  83, 240, 243
400 DATA  62,  20, 187, 250, 111

410 DATA 243,  42, 244, 243, 126
420 DATA  35,  34, 244, 243, 254
430 DATA  31, 250, 111, 243, 254
440 DATA 144, 242, 111, 243, 214
450 DATA  32,   1,   8,   0,  42
460 DATA  54,  92,  36,   9,  61
470 DATA  32, 252,  34, 242, 243
480 DATA 123, 230,  24, 246,  64
490 DATA 103, 123, 230,   7, 183
500 DATA  31,  31,  31,  31, 130

510 DATA 111,  34, 238, 243, 205
520 DATA 113, 243,  58, 241, 243
530 DATA  60,  60,  50, 241, 243
540 DATA 193,  16, 164, 201, 193
550 DATA 201,  17, 206, 243,   6
560 DATA  32,  62,   0,  18,  19
570 DATA  16, 252, 237,  91, 242
580 DATA 243,  33, 206, 243,   6
590 DATA   8, 197,  26,   1,   2
600 DATA   4, 197,  23, 245, 203

610 DATA  22, 241, 203,  22,  16
620 DATA 247,  35, 193,  13,  32
630 DATA 241,  43, 126, 245,  43
640 DATA 126,  35,  35, 119,  35
650 DATA 241, 119,  35,  19, 193
660 DATA  16, 220,  42, 238, 243
670 DATA  17, 206, 243,  14,   2
680 DATA 229,   6,   8,  26, 119
```

```
690 DATA  35,  19,  26, 119,  19
700 DATA  43,  36,  16, 245, 225
710 DATA  62,  32, 133, 111,  48
720 DATA   4,  62,   8, 132, 103
730 DATA  13,  32, 228, 201,   0
740 DATA   0,   0,   0,   0,   0
```

Дисассемблер программы:

```
62200   2A0B5C        LD HL,(5C0BH)  ;См. с.109...111.
62203   010400        LD BC,0004     ;Сдвиг от DEFADD на 4 бай-
62206       09        ADD HL,BC      ;та (см. с.109...111).
62207       56        LD D,(HL)      ;Координата x.
62208   010800        LD BC,0008     ;Сдвиг от DEFADD еще на 8
62211       09        ADD HL,BC      ;байтов (см.с.109...111).
62212       5E        LD E,(HL)      ;Координата y.
62213 ED53F0F3        LD(COOR_Y),DE  ;Переброска параметров y
                                     ;и x в адреса 62448,62449
62217     3E63        LD A,63        ;63H=99DEC - количество
                                     ;символов в сообщении.
62219       47        LD B,A         ;Счетчик символов.
62220   2124F4        LD HL,F424     ;F424H=62500DEC- началь-
                                     ;ный адрес сообщения.
62223   22F4F3        LD (POINT),HL  ;По адресу 62452 создали
                                     ;переменную-указатель на
                                     ;адрес печатаемого символа
62226       C5 LOOP_M PUSH BC        ;Сохранение счетчика на
                                     ;стеке.
62227 ED5BF0F3        LD DE,(COOR_Y) ;Координаты y,x.
62231     3E1E        LD A,1E        ;Предел по x=31.
62233       BA        CP D           ;Сравнили x и 31.
62234   F225F3        JP P,SKIP_1    ;Если все O.K. - переход
                                     ;на SKIP_1, иначе обнуля-
62237     1600        LD D,00        ;ем x и увеличиваем на 2
62239       1C        INC E          ;значение y, т.е. перехо-
62240       1C        INC E          ;дим на новую строку.
62241 ED53F0F3        LD(COOR_Y),DE  ;Запоминаем текущую коор-
                                     ;динату позиции печати.
```

```
62245    3E14 SKIP_1 LD A,14         ;Проверка
62247      BB        CP E           ;на y < 21.
62248   FA6FF3       JP M,EXIT      ;Если y>=21 - выход.
62251   2AF4F3       LD HL,(POINT)  ;Адрес символа, подлежа-
                                    ;щего печати.
62254      7E        LD A,(HL)      ;Взяли этот символ,
62255      23        INC HL         ;наметили следующий
62256   22F4F3       LD (POINT),HL  ;и передвинули указатель.
62259    FE1F        CP 1F          ;Если код символа меньше
62261   FA6FF3       JP M,EXIT      ;32 или больше 143,
62264    FE90        CP 90          ;выполняется выход через
62266   F26FF3       JP P,EXIT      ;метку EXIT.
62269    D620        SUB 20         ;Из кода символа вычита-
                                    ;ется 32.
62271   010800       LD BC,0008     ;На каждый символ расходу-
                                    ;ется 8 байтов.
62274   2A365C       LD HL,(5C36)   ; = 23606 - адрес систем-
                                    ;ной переменной CHARS (см.
                                    ;стр.65).
62277      24        INC H          ;Теперь HL указывает точно
                                    ;на первый символ систем-
                                    ;ного набора.
62278      09 LOOP_C ADD HL,BC      ;Теперь на второй символ и
                                    ;т.д,
62279      3D        DEC A          ;Когда в аккумуляторе бу-
62280    20FC        JR NZ,LOOP_C   ;дет 0, HL укажет точно на
                                    ;нужный нам символ, равный
                                    ;тому, на который в данный
                                    ;момент указывает указа-
                                    ;тель POINT.
62282   22F2F3       LD (CH_ADD),HL ;Запомнили адрес начала его
                                    ;шаблона в ПЗУ компьютера.
62285      7B        LD A,E         ;╓
62286    E618        AND 18         ;╟ Расчет адреса по
62288    F640        OR 40          ;╟ координатам.
62290      67        LD H,A         ;╟
62291      7B        LD A,E         ;╟
62292    E607        AND 07         ;╙
```

```
62294       B7          OR A            ;╟
62295       1F          RRA             ;╟
62296       1F          RRA             ;╟
62297       1F          RRA             ;╟
62298       1F          RRA             ;╟
62299       82          ADD A,D         ;╟
62300       6F          LD L,A          ;╟
62301   22EEF3          LD(ROWCOL),A    ;Эта переменная
                                        ;содержит адрес в дисплей-
                                        ;ном файле, соответствую-
                                        ;щий начальной линии изо-
                                        ;бражаемого символа.
62304   CD71F3          CALL DOUBL      ;Вызов подпрограммы, вы-
                                        ;полняющей непосредственно
                                        ;печать символов двойной
                                        ;ширины и высоты.
62307   3AF1F3          LD A,(COOR_X)   ;Увеличиваем на 2
62310       3C          INC A           ;указатель координаты
62311       3C          INC A           ;по горизонтали
62312   32F1F3          LD (COOR_X),A   ;и запоминаем новую пози
                                        ;цию печати.
62315       C1          POP BC          ;Восстановили счетчик
                                        ;символов.
62316     10A4          DJNZ, LOOP_M    ;Если не все символы напе-
                                        ;чатаны, возврат.
62318       C9          RET             ;Генеральный выход из
                                        ;процедуры.
62319       C1 EXIT     POP BC          ;Перед аварийным выходом
                                        ;очищаем стек.
62320       C9          RET             ;Аварийный выход, если
                                        ;параметр у не в допуске
                                        ;или символ - непечатный.
```

Процедура DOUBLE.

Если до сих пор шли подготовительные операции, то сам алгоритм удвоения изображения символа реализован именно в этой процедуре (мы назвали ее DOUBLE). Основной принцип действия ее

состоит в том, что каждому символу соответствует восьмибайтный шаблон в ПЗУ - на него указывает пара регистров DE. Строка за строкой шаблона (байт за байтом) могут быть скопированы в аккумулятор. В то же время, в оперативной памяти создается временный рабочий буфер размером 32 байта для удвоенного символа. На начало этого буфера указывает регистровая пара HL. Теперь задача сводится к тому, чтобы бит за битом (пиксел за пикселом) перенести рисунок из аккумулятора A в буфер (HL), при этом одному пикселу (биту) в A должны соответствовать два бита в буфере.

Здесь это сделано оригинальным приемом с помощью операций RLA и RL (HL). Причем на одну операцию RLA приходятся две операции RL (HL). Те, кто знакомы с этими операциями по нашей книге "Первые шаги в машинном коде", знают, что все биты смещаются влево. При этом старший (седьмой) бит поступает в флаг C регистра F, а то, что там было перед операцией, отправляется в нулевой бит. Таким образом, через флаг C происходит копирование с удвоением из аккумулятора в буфер. Чтобы во время операции RL (HL) не искажалось содержимое флага C, принятое из аккумулятора, оно запоминается на стеке PUSH AF и восстанавливается, когда ротация завершена - POP AF.

Рассмотрим пример:

	Флаг C	Аккумулятор	Буфер
Исходно:	0	10101010	00000000
		Шаг 1	
RLA	1	01010100	00000000
PUSH AF	1	01010100	00000000
RL (HL)	0	01010100	00000001
POP AF	1	01010100	00000001
RL (HL)	0	01010100	00000011
		Шаг 2	
RLA	0	10101000	00000011
PUSH AF	0	10101000	00000011
RL (HL)	0	10101000	00000110
POP AF	0	10101000	00000110
RL (HL)	0	10101000	00001100

```
                          Шаг 3
  RLA           1         01010000          00001100
  PUSH AF       1         01010000          00001100
  RL (HL)       0         01010000          00011001
  POP AF        1         01010000          00011001
  RL (HL)       0         01010000          00110011
```

И так далее ....

Как видите, четырех шагов достаточно, чтобы скопировать пол-байта шаблона в один байт буфера. Еще четыре шага - и одной строке шаблона будет соответствовать двухбайтная пара в буфере.

Теперь рассмотрим саму процедуру DOUBLE.

```
62321   11CEF3 DOUBL  LD DE,BUFFER   ;Адрес 62414.
62324     0620        LD B,20         ;Счетчик цикла = 32.
62326     3E00        LD A,00         ;
62328       12 LOOP_H LD (DE),A       ;Цикл, обнуляющий
62329       13        INC DE          ;буфер.
62330     10FC        DJNZ FC LOOP_H  ;Конец цикла.
62332 ED5BF2F3        LD DE,(CH_ADD)  ;DE указывает на шаблон
                                      ;текущего печатаемого
                                      ;символа.
62336   21CEF3        LD HL,BUFFER    ;HL указывает на начало
                                      ;буфера.
62339     0608        LD B,08         ;Счетчик по 8.
62341       C5 LOOP_X PUSH BC         ;Запомнили его на стеке.
62342       1A        LD A,(DE)       ;Приняли линию шаблона
                                      ;текущего символа.
62343   010204        LD BC,0402      ;Организовали два счетчика
                                      ;- на 2 шага в регистре C
                                      ;и на 4 шага - в B.
62346       C5 AGAIN  PUSH BC         ;Запомнили их на стеке.
62347       17 LOOP_S RLA             ;Приняли бит аккумулятора
                                      ;во флаг C.
62348       F5        PUSH AF         ;Запомнили флаг C.
62349     CB16        RL (HL)         ;Приняли этот бит из фла-
                                      ;га C - в буфер.
```

```
62351       F1        POP AF          ;Восстановили флаг C
62352     CB16        RL (HL)         ;и приняли его еще раз.
62354     10F7        DJNZ LOOP_S     ;Возврат в начало цикла,
                                      ;пока не сделаем 4 про-
                                      ;хода для заполнения од-
                                      ;ного байта в буфере.
62356       23        INC HL          ;Перешли ко 2-му байту
                                      ;буфера.
62357       C1        POP BC          ;Вновь в регистре B уста-
                                      ;новили счетчик на 4 про-
                                      ;хода,
62358       0D        DEC C           ;но уменьшили счетчик в C.
62359     20F1        JR NZ,AGAIN     ;Возврат, если второй байт
                                      ;буфера еще не заполнен.
```

К этому моменту мы как бы удвоили одну строку шаблона по горизонтали и вместо 8 битов имеем в буфере соответствующие ей 16 битов (2 байта). Но нам надо удвоить эту строку и по вертикали. Поэтому в третий и четвертый байты буфера заполняем копией первого и второго.

```
62361       2B        DEC HL          ;Возвращаемся ко второму
                                      ;байту буфера.
62362       7E        LD A,(HL)       ;Взяли его в аккумулятор
62363       F5        PUSH AF         ;и запомнили на стеке.
62364       2B        DEC HL          ;Возвращаемся к 1-ому
                                      ;байту буфера.
62365       7E        LD A,(HL)       ;Взяли его в аккумулятор.
62366       23        INC HL          ;Переход ко 2-ому байту.
62367       23        INC HL          ;Переход к 3-му байту.
62368       77        LD (HL),A       ;Копирование 1-го байта
                                      ;в третий.
62369       23        INC HL          ;Переход к 4-му байту.
62370       F1        POP AF          ;Восстановление 2-го
                                      ;байта.
62371       77        LD (HL),A       ;Копирование его в
                                      ;четвертый байт.
```

Итак, мы скопировали одну строку шаблона символа в четыре байта буфера. Но у нас есть 8 строк у символа и их тоже надо так же скопировать.

```
62372       23          INC HL          ;Переход к очередному
                                        ;байту буфера.
62373       13          INC DE          ;Переход к очередной
                                        ;строке шаблона.
62374       C1          POP BC          ;Восстановление счет-
                                        ;чика строк шаблона.
62375     10DC          DJNZ LOOP_X     ;Если не все строки еще
                                        ;обработаны, то возврат.
```

Итак, мы скопировали все 8 строк шаблона в 32-байтный буфер. Теперь настало время поместить изображение удвоенного символа на экран, то есть надо скопировать биты из буфера в дисплейный файл (с учетом координат позиции печати). Сама эта операция достаточно тривиальна. Регистровая пара HL является указателем на адрес в дисплейном файле, пара DE - указывает на адрес в буфере. Организуется счетчик цикла в C = 2 по рядам экрана и внутренний в B = 8 по линиям пикселов в ряду. Единственный нюанс - необходимость после печати первого ряда проверить не произошел ли переход через границы текущего сегмента экрана и соответственно отреагировать изменением содержимого указателя в HL.

```
62377   2AEEF3          LD HL,(ROWCOL)  ;Адрес в экранном файле.
62380   11CEF3          LD DE,BUFFER    ;Адрес буфера.
62383     0E02          LD C,02         ;Счетчик рядов экрана.
62385       E5 LOOP_F   PUSH HL
62386     0608          LD B,08         ;Счетчик линий в ряду.
62388       1A LOOP_Y   LD A,(DE)       ;Переброска линии из
62389       77          LD (HL),A       ;буфера на экран (не-
                                        ;четной).
62390       23          INC HL          ;Сдвиг вправо.
62391       13          INC DE
62392       1A          LD A,(DE)       ;Переброска четной линии
62393       77          LD (HL),A       ;из буфера на экран.
```

```
62394       13          INC DE
62395       2B          DEC HL          ;Сдвиг на знакоместо
                                        ;влево.
62396       24          INC H           ;Переход к следующей
                                        ;линии.
62397     10F5          DJNZ LOOP_Y     ;Повтор для 8 пар линий.
62399       E1          POP HL
62400     3E20          LD A,20         ;Переход к следующему
62402       85          ADD A,L         ;ряду знакомест
62403       6F          LD L,A          ;на экране.
62404     3004          JR NC,SKIP_4    ;Проверка не вышли ли
                                        ;за границы текщего эк-
                                        ;ранного сегмента.
62406     3E08          LD A,08         ;Если да, то переход
62408       84          ADD A,H         ;к следующему сегменту
62409       67          LD H,A          ;увеличением H на 8.
62410       0D SKIP_4   DEC C
62411     20E4          JR NZ,LOOP_F    ;Повтор для 2-х рядов.
62413       C9          RET             ;Возврат в вызывающую
                                         процедуру.

62414      BUFFER       DEFM
62446      ROWCOL       DEFW
62448      COOR_Y       DEFB
62449      COOR_X       DEFB
62450      CH_ADD       DEFW
62452      POINT        DEFW
62500      MESSAG       DEFL
```

3.6. Масштабное увеличение текста. Печать по вертикали.

Эта программа работает примерно так же, как и предыдущая, но печать сообщения выполняется по вертикали. Точно также здесь требуется ввести в память компьютера текстовое сообщение, подлежащее выдаче на экран. Это выполняется в строках 220...290 демонстрационной БЕЙСИК-программы.

Назовем эту программу - FN e (x,y). Здесь x,y - координаты исходной позиции печати. Длина всей программы - 215 байтов, начальный адрес - 61900.

Поскольку символы имеют удвоенный размер, в высоту экрана укладываются только 12 символов. Программа не имеет возможности переноса текста и, если Вам надо напечатать сообщение более длинное, то надо ее вызывать несколько раз. Если в Вашем сообщении используется пробел, то вместо него подставьте графический символ, находящийся на клавише "8". Демонстрационная программа использует в своей работе также и ранее рассмотренную нами программу FN c (x,y,h,v,c,b,f) для цветового оформления экрана перед печатью текста.

```
 10 REM *** Загрузчик машинного кода
 20 LET adr=61900: LET long=210: LET z=0
 30 FOR i=0 TO long-1: READ a
 40 POKE (adr+i),a: LET z=z+a
 50 NEXT i
 60 LET z=INT (((z/long)-INT (z/long))*long)
 70 READ a
 80 IF a<>z THEN PRINT "??": STOP
 90 REM *** Пример использования процедуры
100 DEF FN c(x,y,h,v,c,b,f)=USR 62600
110 DEF FN e(x,y)=USR 61900
120 BORDER 1: PAPER 0: INK 0: CLS
130 LET n$="Spectrum"
140 GO SUB 220
```

```
150 LET pap=1
160 FOR k=5 TO 25 STEP 4
170 RANDOMIZE FN c(k-1,2,4,19,pap,0,0)
180 RANDOMIZE FN e(x,3)
190 LET pap=pap+1
200 NEXT k
210 STOP
220 LET m=LEN(n$)
230 LET s=62499
240 FOR k=1 TO m
250 LET n=CODE n$ (k)
260 POKE s+k,n
270 NEXT k
280 POKE s+k,13
290 RETURN

300 REM *** Данные для машинного кода
310 DATA  42,  11,  92,   1,   4
320 DATA   0,   9,  86,   1,   8
330 DATA   0,   9,  94, 237,  83
340 DATA 191, 242,  62,  99,  71
350 DATA  33,  36, 244,  34, 195
360 DATA 242, 197, 237,  91, 191
370 DATA 242,  62,  30, 186, 242
380 DATA 244, 241, 195,  62, 242
390 DATA  62,  20, 187, 250,  62
400 DATA 242,  42, 195, 242, 126

410 DATA  35,  34, 195, 242, 254
420 DATA  31, 250,  62, 242, 254
430 DATA 144, 242,  62, 242, 214
440 DATA  32,   1,   8,   0,  42
450 DATA  54,  92,  36,   9,  61
460 DATA  32, 252,  34, 193, 242
470 DATA 123, 230,  24, 246,  64
480 DATA 103, 123, 230,   7, 183
490 DATA  31,  31,  31,  31, 130
500 DATA 111,  34, 189, 242, 205
```

```
510 DATA  64, 242,  58, 191, 242
520 DATA  60,  60,  50, 191, 242
530 DATA 193,  16, 169, 201, 193
540 DATA 201,  17, 157, 242,   6
550 DATA  32,  62,   0,  18,  19
560 DATA  16, 252, 237,  91, 193
570 DATA 242,  33, 157, 242,   6
580 DATA   8, 197,  26,   1,   2
590 DATA   4, 197,  23, 245, 203
600 DATA  22, 241, 203,  22,  16

610 DATA 247,  35, 193,  13,  32
620 DATA 241,  43, 126, 245,  43
630 DATA 126,  35,  35, 119,  35
640 DATA 241, 119,  35,  19, 193
650 DATA  16, 220,  42, 189, 242
660 DATA  17, 157, 242,  14,   2
670 DATA 229,   6,   8,  26, 119
680 DATA  35,  19,  26, 119,  19
690 DATA  43,  36,  16, 245, 225
700 DATA  62,  32, 133, 111,  48

710 DATA   4,  62,   8, 132, 103
720 DATA  13,  32, 228, 201,   0
730 DATA   0,   0,   0,   0,   0
```

Дисассемблер программы:

```
61900   2A0B5C        LD HL,(5C0BH)  ;См. с. 109...111.
61903   010400        LD BC,0004     ;Сдвиг от DEFADD на 4 бай-
61906       09        ADD HL,BC      ;та (см. с.109...111).
61907       56        LD D,(HL)      ;Координата x.
61908   010800        LD BC,0008     ;Сдвиг от DEFADD еще на 8
61911       09        ADD HL,BC      ;байтов (см.с.109...111).
61912       5E        LD E,(HL)      ;Координата y.
61913 ED53BFF2        LD(COOR_Y),DE  ;Переброска параметров y
                                     ;и x в адреса 62143,62144
61917     3E63        LD A,63        ;63H=99DEC - размер буфе-
                                     ;ра под сообщение.
```

```
61919       47          LD B,A          ;Счетчик символов.
61920   2124F4          LD HL,BUFFER    ;F424H=62500DEC- началь-
                                        ;ный адрес буфера сообще-
                                        ;ния (используется тот же
                                        ;буфер, что и в предыду-
                                        ;щей программе).
61923   22C3F2          LD (POINT),HL   ;По адресу 62147 создали
                                        ;переменную-указатель на
                                        ;адрес печатаемого символа
61926       C5 LOOP_M   PUSH BC         ;Сохранение счетчика на
                                        ;стеке.
61927 ED5BBFF2          LD DE,(COOR_Y)  ;Координаты y,x.
61931     3E1E          LD A,1E         ;Предел по x=31.
61933       BA          CP D            ;Сравнили x и 31.
61934   F2F4F1          JP P,SKIP_1     ;Если все О.К. - переход
                                        ;на SKIP_1,
61937   C33EF2          JP EXIT         ;иначе - аварийный выход.
61940     3E14 SKIP_1   LD A,14         ;Проверка
61942       BB          CP E            ;на y < 21.
61943   FA3EF2          JP M,EXIT       ;Если y>=21 - выход.
61946   2AC3F2          LD HL,(POINT)   ;Адрес символа, подлежа-
                                        ;щего печати.
61949       7E          LD A,(HL)       ;Взяли этот символ,
61950       23          INC HL          ;наметили следующий
61951   22C3F2          LD (POINT),HL   ;и передвинули указатель.
61954     FE1F          CP 1F           ;Если код символа меньше
61956   FA3EF2          JP M,EXIT       ;32 или больше 143,
61959     FE90          CP 90           ;выполняется выход через
61961   F23EF2          JP P,EXIT       ;метку EXIT.
61964     D620          SUB 20          ;Из кода символа вычита-
                                        ;ется 32.
61966   010800          LD BC,0008      ;На каждый символ расходу-
                                        ;ется 8 байтов.
61969   2A365C          LD HL,(5C36)    ; = 23606 - адрес систем-
                                        ;ной переменной CHARS (см.
                                        ;стр. 65).
61972       24          INC H           ;Теперь HL указывает точно
                                        ;на первый символ набора.
```

```
61973       09 LOOP_C ADD HL,BC       ;Теперь на второй символ и
                                      ;т.д,
61974       3D        DEC A           ;Когда в аккумуляторе бу-
61975     20FC        JR NZ,LOOP_C    ;дет 0, HL укажет точно на
                                      ;нужный нам символ, равный
                                      ;тому, на который в данный
                                      ;момент указывает указа-
                                      ;тель POINT.
61977   22C1F2        LD (CH_ADD),HL  ;Запомнили адрес начала
                                      ;его шаблона в ПЗУ компью-
                                      ;тера.
61980       7B        LD A,E          ;╓
61981     E618        AND 18          ;╟ Расчет адреса по
61983     F640        OR 40           ;╟    координатам.
61985       67        LD H,A          ;╟
61986       7B        LD A,E          ;╟
61987     E607        AND 07          ;╟
61989       B7        OR A            ;╟
61990       1F        RRA             ;╟
61991       1F        RRA             ;╟
61992       1F        RRA             ;╟
61993       1F        RRA             ;╟
61994       82        ADD A,D         ;╟
61995       6F        LD L,A          ;╙
61996   22BDF2        LD(ROWCOL),HL   ;Переменная по адресу
                                      ;62141 несет информацию о
                                      ;номере ряда и столбца по-
                                      ;зиции печати.
61999   CD40F2        CALL DOUBL      ;Вызов подпрограммы, вы-
                                      ;полняющей непосредственно
                                      ;печать символов двойной
                                      ;ширины и высоты.
62002   3ABFF2        LD A,(COOR_Y)   ;Увеличиваем на 2
62005       3C        INC A           ;указатель координаты
62006       3C        INC A           ;по вертикали
62007   32BFF2        LD (COOR_Y),A   ;и запоминаем новую пози-
                                      ;цию печати.
62010       C1        POP BC          ;Восстановили счетчик.
```

```
62011     10A9         DJNZ, LOOP_M   ;Если не все символы напе-
                                      ;чатаны, возврат.
62013       C9         RET            ;Генеральный выход из
                                      ;процедуры.
62014       C1 EXIT    POP BC         ;Перед аварийным выходом
                                      ;очищаем стек.
62015       C9         RET            ;Аварийный выход, если
                                      ;параметр у не в допуске
                                      ;или символ - непечатный.

62016   119DF2 DOUBL   LD DE,BUFFER   ;Адрес 62109.
62019     0620         LD B,20        ;Счетчик цикла = 32.
62021     3E00         LD A,00        ;
62023       12 LOOP_H  LD (DE),A      ;Цикл, обнуляющий
62024       13         INC DE         ;буфер.
62025     10FC         DJNZ FC LOOP_H ;Конец цикла.
62027 ED5BC1F2         LD DE,(CH_ADD) ;DE указывает на шаблон
                                      ;текущего печатаемого
                                      ;символа.
62031   219DF2         LD HL,BUFFER   ;HL указывает на начало
                                      ;буфера.
62034     0608         LD B,08        ;Счетчик по 8.
62036       C5 LOOP_X  PUSH BC        ;Запомнили его на стеке.
62037       1A         LD A,(DE)      ;Приняли линию шаблона
                                      ;текущего символа.
62038   010204         LD BC,0402     ;Организовали два счетчика
                                      ;- на 2 шага в регистре С
                                      ;и на 4 шага - в В.
62041       C5 AGAIN   PUSH BC        ;Запомнили их на стеке.
62042       17 LOOP_S  RLA            ;Приняли бит во флаг С.
62043       F5         PUSH AF        ;Запомнили флаг С.
62044     CB16         RL (HL)        ;Приняли этот бит в буфер.
62046       F1         POP AF         ;Восстановили флаг С
62047     CB16         RL (HL)        ;и приняли его еще раз.
62049     10F7         DJNZ LOOP_S    ;Возврат в начало цикла,
                                      ;пока не сделаем 4 про-
                                      ;хода для заполнения од-
                                      ;ного байта в буфере.
```

```
62051       23          INC HL          ;Перешли ко 2-му байту
                                        ;буфера.
62052       C1          POP BC          ;Вновь в регистре B уста-
                                        ;новили счетчик на 4 про-
                                        ;хода,
62053       0D          DEC C           ;но уменьшили счетчик в C.
62054     20F1          JR NZ,AGAIN     ;Возврат, если второй байт
                                        ;буфера еще не заполнен.
62056       2B          DEC HL          ;Возвращаемся ко второму
                                        ;байту буфера.
62057       7E          LD A,(HL)       ;Взяли его в аккумулятор
62058       F5          PUSH AF         ;и запомнили на стеке.
62059       2B          DEC HL          ;Возвращаемся к 1-ому
                                        ;байту буфера.
62060       7E          LD A,(HL)       ;Взяли его в аккумулятор.
62061       23          INC HL          ;Переход ко 2-ому байту.
62062       23          INC HL          ;Переход к 3-му байту.
62063       77          LD (HL),A       ;Копирование 1-го байта
                                        ;в третий.
62064       23          INC HL          ;Переход к 4-му байту.
62065       F1          POP AF          ;Восстановление 2-го
                                        ;байта.
62066       77          LD (HL),A       ;Копирование его в
                                        ;четвертый байт.
62067       23          INC HL          ;Переход к очередному
                                        ;байту буфера.
62068       13          INC DE          ;Переход к очередной
                                        ;строке шаблона.
62069       C1          POP BC          ;Восстановление счет-
                                        ;чика строк шаблона.
62070     10DC          DJNZ LOOP_X     ;Если не все строки еще
                                        ;обработаны, то возврат.
62072   2ABDF2          LD HL,(ROWCOL)  ;Адрес в экранном файле.
62075   119DF2          LD DE,BUFFER    ;Адрес буфера.
62078     0E02          LD C,02         ;Счетчик рядов экрана.
62080       E5 LOOP_F   PUSH HL
62081     0608          LD B,08          ;Счетчик линий в ряду.
62083       1A LOOP_Y   LD A,(DE)        ;Переброска линии из
```

```
62084       77          LD (HL),A       ;буфера на экран.
62085       23          INC HL          ;Сдвиг на знакоместо
                                        ;вправо.
62086       13          INC DE
62087       1A          LD A,(DE)       ;Переброска четной линии
62088       77          LD (HL),A       ;из буфера на экран.
62089       13          INC DE
62090       2B          DEC HL          ;Сдвиг на знакоместо
                                        ;влево.
62091       24          INC H           ;Переход к следующей
                                        ;линии.
62092     10F5          DJNZ LOOP_Y     ;Повтор для 8 пар линий.
62094       E1          POP HL
62095     3E20          LD A,20         ;Переход к следующему
62097       85          ADD A,L         ;ряду знакомест
62098       6F          LD L,A          ;на экране.
62099     3004          JR NC,SKIP_4    ;Проверка не вышли ли
                                        ;за границы сегмента.
62101     3E08          LD A,08         ;Если да, то переход
62103       84          ADD A,H         ;к следующему сегменту
62104       67          LD H,A          ;увеличением H на 8.
62105       0D SKIP_4   DEC C
62106     20E4          JR NZ,LOOP_F    ;Повтор для 2-х рядов.
62108       C9          RET             ;Возврат.

62109      BUFFER       DEFM
62141      ROWCOL       DEFW
62143      COOR_Y       DEFB
62144      COOR_X       DEFB
62145      CH_ADD       DEFW
62147      POINT        DEFW
62500      MESSAG       DEFL
```

3.7. Печать точек на экране.

До сих пор мы манипулировали с экранными образами, координаты которых были заданы в знакоместах, но как Вы знаете, это не единственный способ. Для графики высокого разрешения более естественен способ задания координат - в пикселах. При этом считаем, что экран имеет 256 пикселов по горизонтали (от 0 до 255) и 176 пикселов по вертикали (от 0 до 175). И начинается графика высокого разрешения с привязкой изображения по пикселам - именно с печати на экране обычной точки.

Конечно, в ПЗУ компьютера есть процедура для печати точек в заданных координатах (см. стр. 85) и можно было бы на этом вопросе и не останавливаться, но поскольку построение многих и многих изображений имеет в своей основе печать точек и на этом основана вся векторная графика, в учебных целях необходимо рассмотреть алгоритм работы такой процедуры и дать несколько примеров ее применения.

Мы назовем предлагаемую процедуру FN f(x,y). Здесь x (0...255) - координата пиксела по горизонтали, а y (0...175) - по вертикали. Нам надо помнить, что эти координаты отсчитываются не так, как координаты в знакоместах. Начальной координатой (0,0) является левый нижний угол экрана (а не левый верхний, как было ранее). Соответственно, правый верхний угол экрана имеет координату (255,175). Кроме того, надо также иметь в виду, что когда мы говорим о левом нижнем угле экрана (0,0), то мы не затрагиваем две самые нижние системные строки компьютера, которые лежат еще ниже, чем нижняя строка основного экрана. Фактически координата (0,0) находится над ними.

```
10 REM *** Загрузчик машинного кода
20 LET adr=61500: LET long=60: LET z=0
30 FOR i=0 TO long-1: READ a
40 POKE (adr+i),a: LET z=z+a
50 NEXT i
60 LET z=INT (((z/long)-INT (z/long))*long)
70 READ a
80 IF a<>z THEN PRINT "??": STOP
```

```
  500 REM ***Данные для машинного кода
  510 DATA  42,  11,  92,   1,   4
  520 DATA   0,   9,  86,  14,   8
  530 DATA   9,  94,  62, 175, 147
  540 DATA 216,  95, 167,  31,  55
  550 DATA  31, 167,  31, 171, 230
  560 DATA 248, 171, 103, 122,   7
  570 DATA   7,   7, 171, 230, 199
  580 DATA 171,   7,   7, 111, 122
  590 DATA 230,   7,  71,   4,  62
  600 DATA 254,  15,  16, 253,   6
  610 DATA 255, 168,  71, 126, 176
  620 DATA 119, 201,   0,   0,   0
  630 DATA  24,   0,   0,   0,   0
```

Дисассемблер программы:

```
61500   2A0B5C        LD HL,(5C0BH)  ;См. с.109...111.
61503   010400        LD BC,0004     ;Сдвиг от DEFADD на 4 бай-
61506      09         ADD HL,BC      ;та (см. с.109...111).
61507      56         LD D,(HL)      ;Координата x.
61508     0E08        LD C,08        ;Сдвиг на
61510      09         ADD HL,BC      ;восемь байтов.
61511      5E         LD E,(HL)      ╟
61512     3EAF        LD A,0AFH      ╟  Расчет адреса по
61514      93         SUB E          ╟  координатам.
61515      D8         RET C          ╟
61516      5F         LD E,A         ╟
61517      A7         AND A          ╟
61518      1F         RRA            ╟
61519      37         SCF            ╟
61520      1F         RRA            ╟
61521      A7         AND A          ╟
61522      1F         RRA            ╟
61523      AB         XOR E          ╟
61524     E6F8        AND 0F8H       ╟
61526      AB         XOR E          ╟
61527      67         LD H,A         ╟
```

```
61528       7A        LD A,D          ‖
61529       07        RLCA            ‖
61530       07        RLCA            ‖
61531       07        RLCA            ‖
61532       AB        XOR E           ‖
61533     E6C7        AND C7          ‖
61534       AB        XOR E           ‖
61536       07        RLCA            ‖
61537       07        RLCA            ‖
61538       6F        LD L,A          ‖
```

Теперь мы знаем номер сегмента, ряда, столбца и линии. Все эти данные хранятся в регистре HL. Последнее, что осталось сделать - это вспомнить, что данная линия в своем знакоместе имеет 8 пикселов, а включить нужно только один из них - необходимый. Фактически его номер определяется остатком от деления координаты x на 8, а вычисляется этот остаток - путем маскирования пяти старших битов, но есть еще один нюанс. Дело в том, что координата x изменяется слева направо 0,1,2... 175, а номера битов в байте, соответствующем найденной нами линии идут наоборот 7,6,5.....0. Поэтому нужно сделать преобразование. В аккумулятор вводится число FE (11111110), имеющее 0 в крайней правой позиции, после чего делается вращение этого байта N+1 раз, где N -остаток от деления x на 8.

```
61539       7A        LD A,D          ;Координата x.
61540     E607        AND 07          ;Маскирование.
61542       47        LD B,A          ;Остаток от деления x на 8
61543       04        INC B           ; + 1
61544     3EFE        LD A,FE         ;Ввели байт 1111 1110
61546       0F LOOP   RRCA            ;Вращение вправо столько
61547     10FD        DJNZ,LOOP       ;раз, сколько установлено
                                      ;в регистре B.
61549     06FF        LD B,0FFH       ;Инверсия, чтобы печать
61551       A8        XOR B           ;точки была черным по бе-
                                      ;лому, а не наоборот.
61552       47        LD B,A          ;Запомнили в B.
61553       7E        LD A,(HL)       ;В аккумулятор приняли то,
```

```
                                    ;что уже содержится на эк-
                                    ;ране в нужной линии.
61554       B0        OR B          ;Включаем требуемый бит
61555       77        LD (HL),A     ;Включаем требуемый пик-
                                    ;сел.
61556       C9        RET           ;Выход из процедуры.
```

Примеры использования процедуры.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Продемонстрируем работу приведенной выше процедуры на некоторых примерах. Эти примеры написаны на БЕЙСИКе и вставляются в приведенную выше БЕЙСИК-программу (стр.145, 146) между строками 80 и 500.

"Экспонента".
~~~~~~~~~~~~~~~

```
100  DEF FN f(x,y) = USR 61500
110  BORDER 6: PAPER 6: INK 0: CLS
120  FOR n=1.19 TO 1.80 STEP 0.01
130  FOR x= 0 TO 22 STEP 0.5
140  LET z= INT(x^n)
150  LET y= INT (x*8)
160  RANDOMIZE FN f(z,y)
170  RANDOMIZE FN f(255-z, 168-y)
180  NEXT x
190  NEXT n
```

"Планета"
~~~~~~~~~

```
100  DEF FN f(x,y) = USR 61500
110  BORDER 0: PAPER 0: INK 4: CLS
120  LET r=60: LET xc=127: LET yc=88
130  GO SUB 400
140  LET r=20: LET xc=75: LET yc=100
150  GO SUB 400
160  STOP
```
~~~~~~~~~

```
400  FOR y=-r TO r
410  FOR x1 = INT (SQR (r*r-y*y)
420  FOR x=-x1 TO x1
430  LET n=INT (RND*(1)*x1*2)+1
440  IF n<x1+x THEN RESTORE FN f(x+xc,y+yc)
450  NEXT x
460  NEXT y
470  RETURN
```

Почему в строке 440 использован оператор RESTORE, а не RANDOMIZE, Вам должно быть вполне понятно (см. стр.123).

"Косинусоида"
~~~~~~~~~~~~~~

```
100  DEF FN f(x,y) = USR 61500
110  BORDER 0: PAPER 0: INK 2: CLS
120  FOR j= 240 TO 160 STEP -4
130  FOR m=1 TO 510
140  LET y=INT (90+60*(cos (m*PI+j)))
150  RANDOMIZE FN f(m,y)
160  NEXT m
170  NEXT j
```

"Городской пейзаж".
~~~~~~~~~~~~~~~~~~~

Имея в своем архиве на кассете несколько процедур в машинных кодах (FN b, FN c, FN e и FN f) и загружая их в верхние области памяти компьютера, мы уже можем получать очень приличные графические изображения, способные украсить Ваши программы. В качестве примера рассмотрим программу, изображающую ночной городской пейзаж. Разноцветные небоскребы, иллюминированные яркой неоновой надписью "SINCLAIR", ночное небо - все это несложно исполняется несколькими строками на БЕЙСИКе (загрузчик машинного кода процедур здесь не приводится - предполагается, что это Вы сделаете сами, например через LOAD ""CODE.

Небоскребы изображаются с помощью процедуры закрашивания окна цветом PAPER - FN b. Вертикальная надпись "SINCLAIR"

печатается крупными буквами процедурой FN e. Эффект нагромождения небоскребов достигается закрашиванием цветных "окон" в случайном порядке. При этом небоскребы могут иметь различную высоту, но основание у них - одно за счет того, что начальная координата по высоте y1 образуется посредством вычитания высоты "окна" из 25-ти.

Для изображения вертикальной надписи "SINCLAIR" необходимо, чтобы этот текст был предварительно, до начала работы занесен в память, что и выполняется в строках 110 - 170. Не забудьте в конце этой записи поместить код CHR 13 (ENTER), который явится сигналом конца сообщения.

В строке 210 вызывается процедура печати точек, что служит для изображения звезд на фоне черного неба. Случайные координаты для их печати выбираются в строках 190 и 200.

Полумесяц изображается накладыванием семейства полуокружностей в строках 530...560. Поскольку мы пока не рассматривали машиннокодовых процедур для изображения дуг, эта операция сделана на БЕЙСИКе оператором DRAW.

Эффект пролетающего метеора исполнен в строках 490 - 510 с помощью семейства отрезков прямых. Здесь применен тоже БЕЙСИК. Процедуру для изображения прямых мы рассмотрим только в следующем параграфе.

```
  1  REM ***** Определение процедур
 10  DEF FN b(x,y,h,v,c,b,f)=USR 62800
 20  DEF FN e(x,y)=USR 61900
 30  DEF FN c(x,y,h,v,c,b,f)=USR 62600
 40  DEF FN f(x,y) = USR 61500
100  PAPER 0: INK 7: BORDER 0
105  REM ***** Задание текстового сообщения
110  CLS : LET n$="SINCLAIR"
120  LET l=LEN n$: LET k=62499
125  REM ***** Занесение текста в память
130  FOR i=1 TO l
```

```
140  LET n=CODE n$(i)
150  POKE k+i,n
160  NEXT i
165  REM ***** Занесение кода <ENTER>
170  POKE k+i,13
175  REM ***** Печать звезд в 74-х верхних линиях экрана
180  FOR i=0 TO 298 STEP 2
190  LET x1=INT (255*RND)
200  LET y1=174-(INT(74*RND))
210  RESTORE FN f(x1,y1)
220  NEXT i
230  RANDOMUZE FN c(0,14,25,15,1,0,0)
235  REM ***** Изображение зданий заднего плана
240  FOR i=1 TO 50
250  RANDOMIZE
260  LET h1=2+INT (RND*4)
270  LET y1=10+INT(RND*15)
280  LET v1=25-y1
290  LET x1=INT(RND*27)
300  LET c1=2+INT (RND*6)
310  RANDOMIZE FN c(x1,y1,h1,v1,c1,0,0)
320  NEXT i
325  REM ***** Изображение зданий переднего плана
330  RANDOMIZE FN c(16,9,1,15,4,1,0)
340  RANDOMIZE FN c(17,6,2,18,4,1,1)
350  RANDOMIZE FN c(19,10,1,14,4,1,0)
360  RANDOMIZE FN b(17,6,2,18,1,0,1)
370  RANDOMIZE FN e(17,6)
380  RANDOMIZE FN c(1,11,5,14,2,0,0)
385  REM ****** Изображение окон в зданиях
390  FOR i=12 TO 22 STEP 2
400  RANDOMIZE FN c(2,i,1,1,7,1,0)
410  RANDOMIZE FN c(4,i,1,1,6,1,0)
420  NEXT i
430  RANDOMIZE FN c(23,18,9,7,3,0,0)
440  FOR i=24 TO 30 STEP 2
450  RANDOMIZE FN c(i,19,1,1,4,1,0)
460  RANDOMIZE FN c(i,21,1,1,4,1,0)
```

```
470  NEXT i
475  REM ***** Изображение метеора
480  INK 6: BRIGHT 1
490  FOR i=-6 TO 6 STEP 2
500  PLOT 250,165-i
510  DRAW -70,i-40: NEXT i
520  INK 7
525  REM ****** Изображение полумесяца
530  FOR i= 0 TO 7
540  PLOT 20+i,150
550  DRAW 0,-50,0.8*PI
560  NEXT i
570  PAUSE 0
```

3.8. Изображение линий.

Известный оператор БЕЙСИКа DRAW служит для рисования отрезков прямых и дуг окружностей. Вам известно, что для того, чтобы был нарисован нужный отрезок, предварительно должна быть задана (PLOT) исходная точка и указаны относительные координаты конца отрезка. Задание относительных координат (приращений по горизонтали и вертикали) во многих случаях является не очень удобным, а порой и просто трудоемкой операций, т.к. программисту требуется сначала рассчитать эти приращения. Гораздо удобнее было бы вводить абсолютные экранные координаты конца отрезка, что и позволяет сделать предлагаемая здесь процедура FN g (x,y,p,q).

x,y - координаты начала отрезка (x<256, y<176);
p,q - координаты конца отрезка (p<256, q<176)

Эта процедура работает много быстрее, чем традиционная команда DRAW и быстрее, чем машиннокодовая процедура, содержащаяся в ПЗУ для этой цели (т.к. последняя в своей работе использует операции со встроенным калькулятором, а он ведет свои расчеты тоже при поддержке процедур ПЗУ). Приведенная процедура

выполняет некоторые проверки, исключающие ввод координат, выходящих за пределы экрана, что позволяет практиковать при использовании процедуры метод проб и ошибок без угрозы потерять и БЕЙСИК-программу и процедуру вследствие зависания компьютера. Тем не менее, если введенная Вами координата будет не только выходить за пределы экрана, но при этом еще и выходить более, чем на 255 пикселов, есть вероятность того, что программа может зависнуть.

Для нас процедура рисования линий - это первый шаг в направлении так называемой векторной графики. Многочисленные поклонники программы ELITE знают, как выглядят такие изображения на экране. Прежде, чем заняться разбором машинного кода процедуры, мы в нескольких словах укрупненно рассмотрим логику ее работы.

Построение линии осуществляется точка за точкой, начиная от исходной координаты. Каждый шаг выполняется со смещением не более, чем на один пиксел как по горизонтали, так и по вертикали до тех пор, пока не будет достигнута конечная точка.

1. Сначала вычисляются вектора от начальной точки до конечной. Если и по x и по y приращения положительные, то в специальной переменной SIGN выставляются +1 для x и +1 для y. Если же какое-то из этих смещений отрицательное, то для него выставляется -1 (то есть FF).

2. Проблемой является тот факт, что если одно из приращений больше другого, то его координата должна изменяться на каждом шаге, а для меньшего - не на каждом. Чтобы это было так, кроме SIGN вводится еще контрольная пара BC. В ней тому приращению, которое больше по абсолютной величине, соответствует 01 или FF, а меньшему - 0. И теперь при построении новой точки шаг по координатам берется то из SIGN, то из BC. Чем сильнее по абсолютной величине отличаются приращения по x и y, тем чаще построение новой точки делается с шагом, взятым из BC, когда отрабатывается только одна координата, а вторая остается неизменной. Эта логика организована в адресах 60782 - 60789.

3. В остальном логика работы процедуры достаточно обычна. Отметим только, что сама печать точек по координатам, находящимся в DE выполняется процедурой PLOT (60851), которая аналогична разобранной в предыдущем разделе.

```
10 REM *** Загрузчик машинного кода
20 LET adr=60700: LET long=210: LET z=0
30 FOR i=0 TO long-1: READ a
40 POKE (adr+i),a: LET z=z+a
50 NEXT i
60 LET z=INT (((z/long)-INT (z/long))*long)
70 READ a
80 IF a<>z THEN PRINT "??": STOP
500 REM ***Данные для машинного кода
510 DATA  42,  11,  92,   1,   4
520 DATA   0,   9,  86,  14,   8
530 DATA   9,  94, 237,  83,  26
540 DATA 237, 205, 226, 237,  94
550 DATA  42,  26, 237, 217, 229
560 DATA 217, 237, 115, 175, 237
570 DATA   1,   1,   1, 122, 148
580 DATA 210,  70, 237,   6, 255
590 DATA 237,  68,  87, 123, 149
600 DATA 210,  80, 237,  14, 255

610 DATA 237,  68,  95, 122, 187
620 DATA  48,  10, 106, 237,  67
630 DATA 177, 237, 175,  71, 195
640 DATA 107, 237, 178, 202, 167
650 DATA 237, 107,  90, 237,  67
660 DATA 177, 237,  14,   0,  99
670 DATA 123,  31, 133, 218, 118
680 DATA 237, 188, 218, 128, 237
690 DATA 148,  87, 217, 237,  91
700 DATA 177, 237, 195, 132, 237

710 DATA  87, 197, 217, 209,  42
720 DATA  26, 237, 123, 133,  95
```

```
730 DATA 122,  60, 132, 218, 164
740 DATA 237, 202, 167, 237,  61
750 DATA  87, 237,  83,  26, 237
760 DATA 205, 179, 237, 217, 122
770 DATA  29,  32, 205, 195, 167
780 DATA 237, 202, 147, 237, 237
790 DATA 123, 175, 237, 217, 225
800 DATA 217, 201, 181, 214,   1

810 DATA   1,  62, 175, 147, 218
820 DATA 249,  36,  95, 167,  31
830 DATA  55,  31, 167,  31, 171
840 DATA 230, 248, 171, 103, 122
850 DATA   7,   7,   7, 171, 230
860 DATA 199, 171,   7,   7, 111
870 DATA 122, 230,   7,  71,   4
880 DATA  62, 254,  15,  16, 253
890 DATA   6, 255, 168,  71, 126
900 DATA 176, 119, 201, 229, 197
910 DATA 205, 179, 237, 193, 225
920 DATA   9,  86,   9, 201,   0
930 DATA 192,   0,   0,   0,   0
```

Дисассемблер программы:

```
60698           COORD DEFW          ;Здесь создается програм-
                                    ;мная переменная, хранящая
                                    ;координаты исходной точки
60700   2A0B5C      LD HL,(5C0BH)   ;См. с.109...111.
60703   010400      LD BC,0004      ;Сдвиг от DEFADD на 4 бай-
60706       09      ADD HL,BC       ;та (см. с.109...111).
60707       56      LD D,(HL)       ;Координата x.
60708     0E08      LD C,08         ;Сдвиг на
60710       09      ADD HL,BC       ;восемь байтов.
60711       5E      LD E,(HL)       ;Координата y.
60712 ED531AED      LD(COORD),DE    ;Запомнили x,y
60716   CDE2ED      CALL BEGIN      ;Вызов подпрограммы BEGIN
                                    ;(60898), которая выполнит
```

```
                                    ;печать исходной точки.
60719       5E        LD E,(HL)     ;Ввод координаты q.
60720   2A1AED        LD HL,(ED1A)  ;Приняли координаты x и y.
60723       D9        EXX           ;Сохранение на стеке сос-
60724       E5        PUSH HL       ;тояния альтернативной па-
60725       D9        EXX           ;ры HL, поскольку при
                                    ;работе программы она мо-
                                    ;жет быть коррумпирована.
60726 ED73AFED        LD(STK_P),SP  ;Запомнили состояние ука-
                                    ;зателя стека в соответс-
                                    ;твующей переменной.
60730   010101        LD BC,0101    ;Исходная установка для
                                    ;инициализации переменной
                                    ;SIGN (60849).
60733       7A        LD A,D        ;Параметр p.
60734       94        SUB H         ;Вычли из него параметр x.
60735   D246ED        JP NC,CONT_1  ;Если p>x, то регистр B
                                    ;менять не надо и перехо-
                                    ;дим на метку CONT_1.
60738     06FF        LD B,0FFH     ;В противном случае в B
                                    ;выставляем FF (минус еди-
                                    ;ницу).
60740     ED44        NEG           ;Изменение знака содержи-
                                    ;мого аккумулятора, теперь
                                    ;в нем абсолютная величина
                                    ;приращения по x, а знак -
                                    ;в регистре B.
60742       57 CONT_1 LD D,A        ;Запомнили в D абс.велич.
                                    ;приращения по x.
60743       7B        LD A,E        ;Параметр q.
60744       95        SUB L         ;Вычли из него параметр y.
60745   D250ED        JP NC,CONT_2  ;Если q>y, то регистр C
                                    ;менять не надо и перехо-
                                    ;дим на метку CONT_2.
60748     0EFF        LD C,0FFH     ;В противном случае в C
                                    ;выставляем FF (минус еди-
                                    ;ницу).
```

```
60750     ED44        NEG             ;Изменение знака содержи-
                                      ;мого аккумулятора, теперь
                                      ;в нем абсолютная величина
                                      ;приращения по y, а знак -
                                      ;в регистре C.
60752       5F CONT_2 LD E,A          ;Запомнили в E абс.велич.
                                      ;приращения по y.
60753       7A        LD A,D          ;Сравниваем по абсолютной
60754       BB        CP E            ;величине приращения x и
                                      ;y.
60755     300A        JR NC,CONT_3    ;Если приращение по x
                                      ;больше, то переход на
                                      ;метку CONT_3.
60757       6A        LD L,D          ;Запомнили меньшее.
60758 ED43B1ED        LD (SIGN),BC    ;Запомнили знаки прира-
                                      ;щений в переменной SIGN.
60762       AF        XOR A           ;Обнуление аккумулятора.
60763       47        LD B,A          ;Обнуление B (раз прираще-
                                      ;ние по x - меньшее, то
                                      ;первый шаг по нему будет
                                      ;нулевым.
60764   C36BED        JP CONT_4       ;Переход на CONT_4 для
                                      ;продолжения работы.
```

Сюда мы попадаем только в том случае, если абсолютная величина приращения по x больше, чем приращения по y.

```
60767       B2 CONT_3 OR D            ;Поскольку и в аккумуля-
                                      ;торе и в регистре D на-
                                      ;ходится приращение по x,
                                      ;то эта операция фактиче-
                                      ;ски является проверкой
                                      ;флагов.
60768   CAA7ED        JP Z,FINISH     ;Если приращение уже рав-
                                      ;но нулю, то конец рабо-
                                      ;ты и переход на FINISH.
60771       6B        LD L,E          ;Приращение по y.
60772       5A        LD E,D          ;Приращение по x.
```

```
60773 ED43B1ED        LD (SIGN),BC   ;Запомнили знаки прира-
                                     ;щений в переменной SIGN.
60777     OE00        LD C,00        ;Переинициализация C. (Раз
                                     ;приращение по y меньше,
                                     ;то первый шаг по нему
                                     ;будет нулевым.
```

В итоге манипуляций, проведенных в строках 60753 - 60777 мы имеем в регистре E то приращение (из x и y), которое имеет большую абсолютную величину, назовем ее - max a в регистре L - то, которое имеет меньшую величину, назовем ее min.

```
60779       63 CONT_4 LD H,E         ;Большее из приращений.
60780       7B        LD A,E         ;Большее из приращений.
60781       1F        RRA            ;Деление пополам.
60782       85 REPEAT ADD A,L        ;0.5max+min
60783   DA76ED        JP C,CONT_5    ;Здесь переполнение воз-
                                     ;можно только если
                                     ;min>0.5max.
60786       BC        CP H           ;Здесь переполнение воз-
60787   DA80ED        JP C,CONT_6    ;можно только если
                                     ;min<0.5max.
60790       94 CONT_5 SUB H          ;Получили min-0.5max
60791       57        LD D,A         ;Запомнили в D
60792       D9        EXX            ;Переход на альтернативный
                                     ;набор регистров, чтобы не
                                     ;портить DE.
60793 ED5BB1ED        LD DE,(SIGN)   ;В DE - знаки приращений
                                     ;по x и y.
60797   C384ED        JP CONT_7

60800       57 CONT_6 LD D,A         ;Запомнили в D min+0.5 max
60801       C5        PUSH BC        ;Переброска через стек
60802       D9        EXX            ;знака приращений из BC в
60803       D1        POP DE         ;альтернативную пару DE.
60804   2A1AED CONT_7 LD HL,(COORD)  ;Текущие координаты пози-
                                     ;ции печати очередной
                                     ;точки.
```

```
60807       7B         LD A,E          ;Знак приращения по y.
60808       85         ADD A,L         ;Переход к новой точке по
                                       ;вертикали.
60809       5F         LD E,A          ;Запомнили ее.
60810       7A         LD A,D          ;Знак приращения по x.
60811       3C         INC A           ;Проверка на возможность
60812       84         ADD A,H         ;выхода за пределы экра-
60813   DAA4ED         JP C,CONT_8     ;на.
60816   CAA7ED         JP Z,FINISH     ;Подготовка к выходу.
60819       3D CONT_9  DEC A           ;Восстановили A после
                                       ;вышеуказанной проверки
60820       57         LD D,A          ;и запомнили в D.
60821 ED531AED         LD (COORD),DE   ;Запомнили новую текущую
                                       ;координату позиции пе-
                                       ;чати точки.
60825   CDB3ED         CALL PLOT       ;Печать точки
60828       D9         EXX             ;Возврат к основному на-
                                       ;бору регистров.
60829       7A         LD A,D
60830       1D         DEC E           ;Уменьшение того прираще-
                                       ;ния, которое было макси-
                                       ;мальным.
60831     20CD         JR NZ,REPEAT    ;Если оно не 0, то нужно
                                       ;работать дальше.
60833   C3A7ED         JP FINISH       ;В противном случае рабо-
                                       ;та завершается.
60836   CA93ED CONT_8  JP Z,CONT_9     ;Предел по x - близок, но
                                       ;еще одну точку напечатать
                                       ;можно - переход на CONT_9
60839 ED7BAFED FINISH  LD SP,(EDAF)    ;Восстановление указателя
                                       ;стека.
60843       D9         EXX             ;Восстановление состояния
60844       E1         POP HL          ;альтернативной пары HL
60845       D9         EXX             ;перед выходом.
60846       C9         RET             ;Генеральный выход.
```

```
60847           STK_P  DEFW          ;В этой программной пере-
                                     ;менной организуется хра-
                                     ;нение значения указателя
                                     ;машинного стека - SP.
60849           SIGN   DEFW          ;Два байта этой переменной
                                     ;выполняют роль знаков
                                     ;приращения координаты.
                                     ;Если x<p,то рисование ли-
                                     ;нии идет слева направо с
                                     ;увеличением координаты и
                                     ;первый байт равен +1. Ес-
                                     ;ли же x>p, то он равен FF
                                     ;или -1. Второй байт уста-
                                     ;навливается в +1 или в FF
                                     ;в зависимости от соотно-
                                     ;шения q и y.
```

Процедура PLOT служит для печати на экране тех точек, из которых состоит наш отрезок. Поскольку она практически соответствует рассмотренной ранее процедуре FN f(x,y), мы не будем подробно рассматривать логику ее работы.

При вызове этой процедуры в регистре D выставлена координата x предполагаемой позиции печати, а в регистре E - координата y.

```
60851     3EAF PLOT   LD A,0AFH      ;AF=175
60853       93        SUB E          ;Проверка не выходит ли за
                                     ;пределы допуска (175) ко-
                                     ;ордината y.
60854   DAF924        JP C,24F9      ;Если так, то переход в
                                     ;ПЗУ на адрес 9465, где
                                     ;записан вызов процедуры
                                     ;обработки ошибок RST 08
                                     ;с кодом перехвата 0A, что
                                     ;дает ошибку "Integer out
                                     ;of range".
60857       5F        LD E,A         ;Дополнение y до 175.
60858       A7        AND A          ;Сброс флага переноса.
```

```
60859       1F          RRA             ;Ротация вправо.
60860       37          SCF             ;Установка флага переноса
60861       1F          RRA             ;Ротация вправо.
60862       A7          AND A           ;Сброс флага переноса.
60863       1F          RRA             ;Ротация вправо.
60864       AB          XOR E           ;Комплексная
60865     E6F8          AND F8          ;операция замещения
60867       AB          XOR E           ;битов 0,1 и 2.
60868       67          LD H,A          ;Сформировали старший байт
                                        ;адреса в дисплейном файле
60869       7A          LD A,D          ;Координата x
60870       07          RLCA            ;Вращение влево.
60871       07          RLCA            ;Вращение влево.
60872       07          RLCA            ;Вращение влево.
60873       AB          XOR E           ;Операция замещения
60874     E6C7          AND C7          ;для битов 3,4,5.
60876       AB          XOR E           ;
60877       07          RLCA            ;Вращение влево.
60878       07          RLCA            ;Вращение влево.
60879       6F          LD L,A          ;Сформировали младший байт
                                        ;адреса в дисплейном файле
60880       7A          LD A,D          ;Координата x.
60881     E607          AND 07H         ;Маскирование.
60883       47          LD B,A          ;Инициализация счетчика
60884       04          INC B           ;оборотов в регистре B.
60885     3EFE          LD A,0FEH       ;Байт FE=254 интересен
                                        ;тем, что все биты, кро-
                                        ;ме одного равны едини-
                                        ;це. Вот эту "дырку" мы
                                        ;и вращаем, пока она не
                                        ;встанет на свое место.
60887       0F AGAIN    RRCA            ;Вращение вправо.
60888     10FD          DJNZ AGAIN      ;Повтор вращения.
60890     06FF          LD B,0FFH       ;Инверсия, чтобы печать
60892       A8          XOR B           ;точки была черным по бе-
                                        ;лому, а не наоборот.
60893       47          LD B,A          ;Запомнили в B.
```

```
60894       7E          LD A,(HL)       ;В аккумулятор приняли то,
                                        ;что уже содержится на эк-
                                        ;ране в нужной линии.
60895       B0          OR B            ;Включаем требуемый бит
60896       77          LD (HL),A       ;Включаем требуемый пик-
                                        ;сел.
60897       C9          RET             ;Выход из процедуры.

60898       E5 BEGIN    PUSH HL         ;Сохранение на стеке
60899       C5          PUSH BC         ;регистров HL и BC.
60900    CDB3ED         CALL PLOT       ;Вызов процедуры PLOT
                                        ; (60851)
60903       C1          POP BC          ;Восстановление со стека
60904       E1          POP HL          ;регистров HL и BC.
60905       09          ADD HL,BC       ;Сдвиг от DEFADD на 8 бай-
                                        ;тов.
60906       56          LD D(HL)        ;Ввод координаты p.
60907       09          ADD HL,BC       ;Сдвиг от DEFADD на 8 бай-
                                        ;тов.
60910       C9          RET             ;Возврат в вызывающую про-
                                        ;грамму.
```

Примеры использования процедуры.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Продемонстрируем работу приведенной выше процедуры на некоторых примерах. Эти примеры написаны на БЕЙСИКе и вставляются в приведенную выше БЕЙСИК-программу (стр. 154) между строками 80 и 500.

"Куб".
~~~~~~~

```
100  DEF FN g(x,y,p,q) = USR 60700
110  BORDER 1: PAPER 1: INK 6
120  CLS
130  FOR j=0 TO 116 STEP 8
140  RANDOMIZE FN g(56,24+j,168,24+j)
150  RANDOMIZE FN g(56+j,24,56+j,136)
160  RANDOMIZE FN g(56+j,136,96+j,160)
```

```
170  RANDOMIZE FN g(168,24+j,208,48+j)
180  NEXT j
190  RANDOMIZE FN g(96,160,208,160)
200  RANDOMIZE FN g(208,160,208,48)
```

"Закат".
~~~~~~~~~~

Эта программа использует также процедуру FN c(), служащую для окраски части экрана цветом PAPER. Машинный код, соответствующий этой процедуре предварительно должен быть подгружен в отведенную для него область памяти.

```
100  DEF FN c(x,y,h,v,c,b,f)=USR 62600
110  DEF FN g(x,y,p,q) = USR 60700
120  BORDER 0: PAPER 1: INK 6
130  CLS
140  RANDOMIZE FN c(0,0,32,17,2,0,0)
150  FOR j=40 TO 174 STEP 12
160  RANDOMIZE FN g(0,j,128,40)
170  RANDOMIZE FN g(255,j,128,40)
180  NEXT j
190  FOR j=6 TO 255 STEP 12
200  RANDOMIZE FN g(j,175,128,40)
210  NEXT j
220  FOR j=36 TO 0 STEP -3
230  RANDOMIZE FN g((10+j*3),j,(248-j*3),j)
240  NEXT j
```

"Пирамиды".
~~~~~~~~~~~~

Процедура, изображающая отрезки прямых, очень хорошо подходит для изображения интерференционных узоров. Они образуются, когда смещения семейства отрезков настолько малы, что в результате не образуются ни отдельные линии, ни сплошная окрашенная поверхность. Приведенная программа показывает, как это происходит и демонстрирует возможность получения очаровательного пейзажа в восточном стиле. Каждая пирамида рисуется вызовом

подпрограммы из строки 400. Интерференционная картина наблюдается на гранях пирамид, что создает эффект трехмерной графики.

```
100  DEF FN b(x,y,h,v,c,b,f)=USR 62800
110  DEF FN c(x,y,h,v,c,b,f)=USR 62600
120  DEF FN g(x,y,p,q) = USR 60700
130  BORDER 0: PAPER 1: INK 2
140  CLS
150  RANDOMIZE FN c(0,8,31,14,6,0,0)
160  LET tx=80: LET ty=136: LET by=24: LET a=16: LET b=128
170  GO SUB 400
180  LET tx=156: LET by=60: LET a=102: LET b=182
190  GO SUB 400: STOP

400  FOR x=a TO b STEP 2
410  RANDOMIZE  FN g(tx,ty,x,by)
420  NEXT x
430  FOR x=b TO b+16
440  RANDOMIZE  FN g(tx,ty,x,by)
450  LET by=by+1
460  NEXT x: RETURN
```

"Плетенка".
~~~~~~~~~~~~

Еще один интересный и до некоторой степени парадоксальный эффект можно получить изображением семейства прямых с малым смещением от линии к линии. Так, при движении одного из концов отрезков нескольких прямых в точках пересечения образуются довольно сложные кривые линии, а штриховой характер изображения придает им сходство со сложными трехмерными поверхностями.

```
100  DEF FN g(x,y,p,q) = USR 60700
110  BORDER 0: PAPER 6: INK 0
120  CLS
130  FOR i=14 TO 2 STEP -1
140  FOR j=0 TO 150 STEP i
150  RANDOMIZE  FN g(50+j,0,j,175)
```
~~~~~~~~~~~~

```
160  RANDOMIZE  FN g(255,j,j,0)
170  RANDOMIZE  FN g(50+j,150,255,150-j)
180  RANDOMIZE  FN g(50+j,150,j,0)
190  NEXT j
200  PAUSE 0: CLS
210  NEXT i
```

"Интерференция".
~~~~~~~~~~~~~~~~

Интересные интерференционные картины на экране могут быть получены простым увеличением количества печатаемых отрезков прямых. Попробуйте поэкспериментировать с приведенной ниже программой, меняя параметр цикла 255 в строке 140.

```
100  DEF FN g(x,y,p,q) = USR 60700
110  BORDER 0: PAPER 1: INK 6
120  CLS
130  FOR i=16 TO 6 STEP -1
140  FOR j=0 TO 255 STEP i
150  RANDOMIZE  FN g(0,0,j,175)
160  RANDOMIZE  FN g(255,175,j,0)
170  RANDOMIZE  FN g(255,0,j,175)
180  RANDOMIZE  FN g(0,175,j,0)
190  NEXT j
200  PAUSE 0: CLS
210  NEXT i
```

3.9. Изображение прямоугольников.

Изображение прямоугольника в БЕЙСИКе выполняется одной командой PLOT и четырьмя командами DRAW. Например:

```
10 PLOT 100,100
20 DRAW 50,0
30 DRAW 0,50
40 DRAW -50,0
50 DRAW 0,50
```
~~~~~~~~~~~~~~~~

Точно так же изображением четырех линий мы можем получить прямоугольник из машинного кода. В общем блоке рассматриваемых нами процедур эта процедура (назовем ее FN h(x,y,h,v)) имеет ту отличительную особенность, что не является абсолютно автономной и при работе вызывает процедуру изображения отрезков прямых FN g(x,y,p,q). Эта процедура вызывается по адресу 60723, и это предполагает, что координаты исходной точки установлены в HL, а координаты точки назначения (абсолютные) - в DE.

Параметры: x,y - координаты левого верхнего угла;
h - ширина
v - высота прямоугольника.

```
10 REM *** Загрузчик машинного кода
20 LET adr=60400: LET long=105: LET z=0
30 FOR i=0 TO long-1: READ a
40 POKE (adr+i),a: LET z=z+a
50 NEXT i
60 LET z=INT (((z/long)-INT (z/long))*long)
70 READ a
80 IF a<>z THEN PRINT "??": STOP

500 REM ***Данные для машинного кода
510 DATA  42,  11,  92,   1,   4
520 DATA   0,   9,  86,  14,   8
530 DATA   9,  94, 237,  83,  82
540 DATA 236,   9,  86,   9,  94
550 DATA 237,  83,  84, 236, 237
560 DATA  91,  82, 236,  58,  85
570 DATA 236, 130, 103,  50,  87
580 DATA 236, 107,  34,  26, 237
590 DATA 205,  51, 237, 237,  91
600 DATA  82, 236,  58,  84, 236

610 DATA 131,  50,  86, 236, 111
620 DATA  98,  34,  26, 237, 205
630 DATA  51, 237,  42,  86, 236
640 DATA  58,  83, 236,  87,  58
```

```
650 DATA  86, 236,  95,  34,  26
660 DATA 237, 205,  51, 237,  42
670 DATA  86, 236,  58,  82, 236
680 DATA  95,  44,  58,  87, 236
690 DATA  87,  34,  26, 237, 205
700 DATA  51, 237, 201,  30,   3
710 DATA  60,  60,  90,  63,   0
720 DATA  86,   0,   0,   0,   0
```

Дисассемблер программы:

```
60400   2A0B5C        LD HL,(5C0BH)  ;См. с. 109...111
60403   010400        LD BC,0004     ;Сдвиг от DEFADD на 4 бай-
60406       09        ADD HL,BC      ;та (см. с.109...111).
60407       56        LD D,(HL)      ;Координата x.
60408     0E08        LD C,08        ;Сдвиг на
60410       09        ADD HL,BC      ;восемь байтов.
60411       5E        LD E,(HL)      ;Координата y.
60412 ED5352EC        LD(Y_UP),DE    ;Запомнили x,y.
60416       09        ADD HL,BC      ;Сдвиг на 8 байтов.
60417       56        LD D,(HL)      ;Параметр h.
60418       09        ADD HL,BC      ;Сдвиг на 8 байтов.
60419       5E        LD E,(HL)      ;Параметр v.
60420 ED5354EC        LD(HIGHT),DE   ;Запомнили h,v.
60424 ED5B52EC        LD DE,(Y_UP)   ;Параметры x,y
60428   3A55EC        LD A,(WIDTH)   ;Приняли ширину пр-ка.
60431       82        ADD A,D        ;Вычисляем x+h и
60432       67        LD H,A         ;запоминаем в регистре H
60433   3257EC        LD (X_RGHT),A  ;и в переменной 60503.
60436       6B        LD L,E         ;Параметр y.
60437   221AED        LD (COORD),HL  ;Запоминаем координаты
                                     ;левого верхего угла в
                                     ;ячейке 60698, откуда они
                                     ;могут быть использованы
                                     ;процедурой FN g() для
                                     ;рисования верхней сторо-
                                     ;ны прямоугольника.
```

```
60440   CD33ED        CALL ED33       ;Вызов процедуры FN g()
                                      ;осуществляется по
                                      ;адресу 60723. Рисуем вер-
                                      ;хнюю сторону.
60443 ED5B52EC        LD DE,(Y_UP)    ;Параметры x,y.
60447   3A54EC        LD A,(HIGHT)    ;Высота пр-ка.
60450       83        ADD A,E         ;Определяем y+v и запоми-
60451   3256EC        LD (Y_DOWN),A   ;наем в переменной 60502
60454       6F        LD L,A          ;и в регистре L.
60455       62        LD H,D          ;Параметр x.
60456   221AED        LD (COORD),HL   ;Запоминаем координаты
                                      ;левого нижнего угла в
                                      ;ячейке 60698.
60459   CD33ED        CALL ED33       ;Рисуем левую сторону.
60462   2A56EC        LD HL,(Y_DOWN)  ;Координаты правого ниж-
                                      ;него угла.
60465   3A53EC        LD A,(X_LEFT)   ;Коорд. x левая.
60468       57        LD D,A
60469   3A56EC        LD A,(Y_DOWN)   ;Коорд. y нижняя
60472       5F        LD E,A
60473   221AED        LD (COORD),HL   ;Правый нижний угол.
60476   CD33ED        CALL ED33       ;Рисуем нижнюю сторону.
60479   2A56EC        LD HL,(Y_DOWN)  ;Координаты правого ниж-
                                      ;него угла.
60482   3A52EC        LD A,(Y_UP)     ;Коорд. y верхней стороны.
60485       5F        LD E,A
60486       2C        INC L
60487   3A57EC        LD A,(X_RGHT)   ;Коорд. x правой стороны.
60490       57        LD D,A
60491   221AED        LD (COORD),HL   ;Правый нижний угол.
60494   CD33ED        CALL ED33       ;Рисуем правую сторону.
60497       C9        RET             ;Выход.
60498          Y_UP   DEFB            ;Координата y верхней
                                      ;стороны.
60499          X_LEFT DEFB            ;Координата x левой
                                      ;стороны.
60500          HIGHT  DEFB            ;Высота прямоугольника.
60501          WIDTH  DEFB            ;Ширина прямоугольника.
```

```
60502          Y_DOWN DEFB          ;Координата y нижней
                                    ;стороны.
60503          X_RGHT DEFB          ;Координата x правой
                                    ;стороны.
60698          COORD  DEFW          ;Переменная, служащая для
                                    ;связи с процедурой печа-
                                    ;ти прямых линий.
```

Примеры использования процедуры.

Продемонстрируем работу приведенной выше процедуры на некоторых примерах. Эти примеры написаны на БЕЙСИКе и вставляются в приведенную выше БЕЙСИК-программу (стр. 166) между строками 80 и 500.

"Каменный цветок"

```
100  DEF FN h(x,y,h,v) = USR 60400
110  BORDER 3: PAPER 3: INK 6: CLS
120  FOR j=1 TO 43
130  RANDOMIZE FN h(j*2,j*3,j,j)
140  RANDOMIZE FN h(j*4,j*3,j,j)
150  RANDOMIZE FN h(j*4,j,j,j)
160  RANDOMIZE FN h(j*2,j,j,j)
170  NEXT j
```

Еще одну разновидность геометрических построений можно получить применением следующей программы:

```
100  DEF FN h(x,y,h,v) = USR 60400
110  BORDER 0: PAPER 2: INK 6 :CLS
120  FOR j=5 TO 38 STEP 2
130  RANDOMIZE FN h(j*2,j,50,50)
140  RANDOMIZE FN h(j*2,130-j,50,50)
150  RANDOMIZE FN h(206-j*2,130-j,50,50)
160  RANDOMIZE FN h(206-j*2,j,50,50)
170  NEXT j
```

3.10. Изображение треугольников.

В предыдущей программе мы использовали при вычислении ко-координат прямоугольника тот факт, что его стороны параллельны. Очередной шаг вперед для изображения произвольных многоугольников мы сделаем, если рассмотрим прием построения треугольника. Для этого служит приводимая здесь процедура FN i(x,y,p,q,r,s).

x,y - координаты первой вершины;
p,g - координаты второй вершины;
r,s - координаты третей вершины треугольника.

Как Вы увидите, для работы этой процедуры создаются три программных переменных PLOT_1...PLOT_3, хранящие координаты вершин треугольника. Несложно развить этот прием и для любых других многоугольных геометрических форм. Как и программа построения прямоугольников, эта процедура использует вызов процедуры FN g(), строящей отрезки прямых. Этот вызов происходит по адресу 60723, для чего координаты исходной точки должны быть установлены в HL, а координаты точки назначения (абсолютные) - в DE. Для обмена между процедурами FN i() и FN g() служит переменная COORD, расположенная по адресу 60698.

```
10 REM *** Загрузчик машинного кода
20 LET adr=60300: LET long=75: LET z=0
30 FOR i=0 TO long-1: READ a
40 POKE (adr+i),a: LET z=z+a
50 NEXT i
60 LET z=INT (((z/long)-INT (z/long))*long)
70 READ a
80 IF a<>z THEN PRINT "??": STOP

500 REM ***Данные для машинного кода
510 DATA  42,  11,  92,   1,   4
520 DATA   0,   9,  86,  14,   8
530 DATA   9,  94, 237,  83, 208
540 DATA 235,   9,  86,   9,  94
550 DATA 237,  83, 210, 235,   9
```

```
560 DATA  86,   9,  94, 237,  83
570 DATA 212, 235,  42, 210, 235
580 DATA  34,  26, 237, 205,  51
590 DATA 237, 237,  91, 208, 235
600 DATA  42, 212, 235,  34,  26

610 DATA 237, 205,  51, 237, 237
620 DATA  91, 210, 235,  42, 208
630 DATA 235,  34,  26, 237, 205
640 DATA  51, 237, 201,  40,   6
650 DATA  40, 143,  80, 123,   0
660 DATA  68,   0,   0,   0,   0
```

Дисассемблер программы:

```
60300   2A0B5C        LD HL,(5C0BH)  ;См. с. 109...111
60303   010400        LD BC,0004     ;Сдвиг от DEFADD на 4 бай-
60306       09        ADD HL,BC      ;та (см. с.109...111).
60307       56        LD D,(HL)      ;Координата x.
60308     0E08        LD C,08        ;Сдвиг на
60310       09        ADD HL,BC      ;восемь байтов.
60311       5E        LD E,(HL)      ;Координата y.
60312 ED53D0EB        LD(PLOT_1),DE  ;Запомнили x,y.
60316       09        ADD HL,BC      ;Сдвиг на 8 байтов.
60317       56        LD D,(HL)      ;Параметр p.
60318       09        ADD HL,BC      ;Сдвиг на 8 байтов.
60319       5E        LD E,(HL)      ;Параметр q.
60320 ED53D2EB        LD(PLOT_2),DE  ;Запомнили p,q.
60324       09        ADD HL,BC      ;Сдвиг на 8 байтов.
60325       56        LD D,(HL)      ;Параметр r.
60326       09        ADD HL,BC      ;Сдвиг на 8 байтов.
60327       5E        LD E,(HL)      ;Параметр s.
60328 ED53D4EB        LD(PLOT_3),DE  ;Запомнили r,s.
60332   2AD2EB        LD HL,(PLOT_2) ;Коорд. 2-ой вершины.
60335   221AED        LD (COORD),HL  ;Запоминаем координаты
                                     ;угла в ячейке 60698.
60338   CD33ED        CALL ED33      ;Вызов процедуры FN g().
                                     ;Рисуем сторону 2-3.
```

```
60341 ED5BD0EB        LD DE,(PLOT_1) ;Координаты 1-ой вершины.
60345   2AD4EB        LD HL,(PLOT_3) ;Координаты 3-ей вершины.
60348   221AED        LD (COORD),HL  ;
60351   CD33ED        CALL ED33      ;Рисование стороны 1-3.
60354 ED5BD2EB        LD DE,(PLOT_2) ;Координаты 2-ой вершины.
60358   2AD0EB        LD HL,(PLOT_1) ;Координаты 1-ой вершины.
60361   221AED        LD (ED1A),HL   ;
60364   CD33ED        CALL ED33      ;Рисование стороны 1-2.
60367       C9        RET            ;Выход из процедуры.

60368          PLOT_1 DEFW           ;Коорд. первой вершины
60370          PLOT_2 DEFW           ;Коорд. второй вершины
60372          PLOT_3 DEFW           ;Коорд. третьей вершины

60698          COORD  DEFW           ;Переменная, служащая для
                                     ;связи с процедурой печа-
                                     ;ти прямых линий.
```

Примеры использования процедуры.

Возможность получения изображения треугольника является очень важной для программиста. К сожалению, в стандартном БЕЙСИКе оператора для этой цели не предусмотрено, а ведь треугольник служит базовым элементом для многих и многих геометрических форм и прочих изображений - пирамиды, горы, деревья, кустарники и пр. и пр. Список можно было бы продолжить. Простейшая программа "Прожектор" покажет Вам каких оригинальных эффектов можно достичь изображением только треугольников.

"Прожектор".

```
     100  DEF FN i(x,y,p,q,r,s) = USR 60300
     110  BORDER 0: PAPER 0: INK 6:
     120  CLS
     130  LET s=4: LET a=0: LET ad=s*PI/128
     140  LET x1=162: LET y1 = 20
     150  FOR i=0 TO 255 STEP s
     160  LET  x=x1+INT (90*SIN a)
```

```
170  LET  y=y1+INT (10*COS a)
180  LET  x2=x1+INT (90*SIN (a+PI))
190  RANDOMIZE  FN i(x,y,x2,y,5,170)
200  LET a=a+ad
210  NEXT i
```

"Калейдоскоп".
~~~~~~~~~~~~~~~~

Если Вы в работе используете цветной телевизор (монитор), то не пожалеете, что набрали следующую программу, но не забудьте, что предварительно должен быть подгружен машинный код процедуры FN b(x,y,h,v,c,b,f).

```
100  DEF FN b(x,y,h,v,c,b,f) = USR 62800
110  DEF FN i(x,y,p,q,r,s) = USR 60300
120  BORDER 4: PAPER 4: INK 1:
130  CLS
140  FOR a=0 TO 75 STEP 4
150  RANDOMIZE FN i(a,0,0,175-a,0,0)
160  RANDOMIZE FN i(a,175,0,a,0,175)
170  RANDOMIZE FN i(255-a,175,255,a,255,175)
180  RANDOMIZE FN i(255-a,0,255,175-a,255,0)
190  NEXT a
200  RANDOMUZE FN b(8,4,16,14,2,0,0)
210  FOR a= 78 TO 178  STEP 4
220  RANDOMIZE FN i(a,38,178,a-40,256-a,138)
230  RANDOMIZE FN i(256-a,138,78,216-a,a,38)
240  NEXT a
```

3.11. Закрашивание области.

Закрашивание произвольной области экрана - довольно часто встречающаяся задача. Нелишне напомнить и о том, что она требует серьезных затрат машинного времени.

Здесь речь идет о том, что если на экране есть некоторое замкнутое изображение (контур), то закрасить его установленным
~~~~~~~~~~~~~~~~

цветом INK - это значит включить все выключенные пикселы, которые внутри этого контура могут быть найдены, оставив без изменения ранее включенные. Может возникнуть вопрос: "А каким же цветом будет заполнен наш контур?" - Это зависит от того, какие атрибуты установлены для того или иного знакоместа. То есть при включении пикселов на экране они приобретут тот цвет, который ранее соответствовал параметру INK для данного знакоместа.

Назовем эту программу FN j(x,y), где x и y - абсолютные координаты исходной точки, с которой начнется заполнение. Разумеется x<255, а y<175. Программа работает по методу "лесного пожара". В окресностях исходной точки включаются все невключенные пикселы (слева, сверху, справа, снизу) и их координаты запоминаются в специально отведенном для этой цели буфере, затем следует переход к первой из новых включенных точек. Окрестности новой точки тоже включаются и запоминаются и следует переход ко второй включенной точке и т.д.

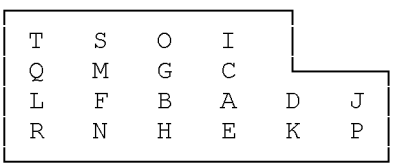


Рис. 18

Если мы начали работу с точки A, то включение точек идет так, как показано на рис. 18.

Важную роль играет в этой процедуре буфер. Он имеет оригинальную организацию. Вы, очевидно, знаете один из приемов организации памяти, называемый стеком. На стек данные закладываются сверху и сверху же и снимаются по принципу "последним пришел - первым уйди". Здесь буфер имеет другую организацию - конвейерную. Конвейер тоже, как и стек, заполняется сверху, но опорожняется снизу. Действует принцип "первым пришел, первым уйди". Главное отличие состоит в том, что у стека динамической явля-

ется только верхняя граница - она постоянно "дышит" вверх или вниз. Основание стека - фиксировано и потому для обслуживания стека в программах достаточно одной переменной - указателя на текущий адрес вершины стека. У конвейера же динамичными оказываются обе границы - и нижняя и верхняя, а потому надо иметь две переменных-указателя. Зато они не "дышат" вверх-вниз, а идут в одном направлении - только вверх. Так, при обслуживании точки A буфер будет иметь вид, показанный на рис. 19 а , а при обслуживании точки C - вид, показанный на рис. 19 б.

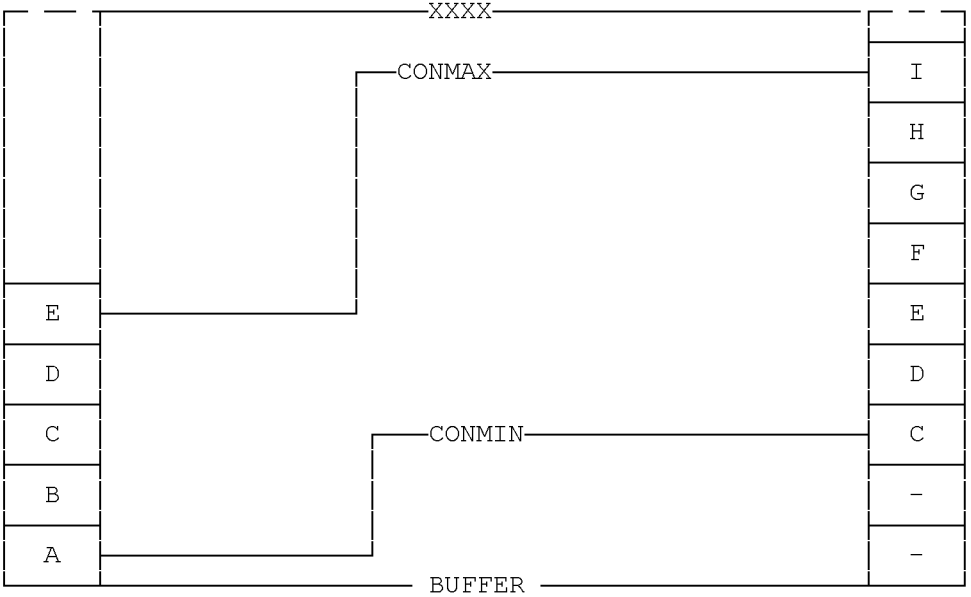


Рис.19а Рис.19б

Каждая клетка на этих рисунках - это два байта памяти, хранящие координаты x и y для данной точки. (Клетки F,G,H добавились при обслуживании т.B, а клетка I - при обслуживании C).

Программная переменная BUFFER содержит адрес физической нижней границы буфера. Он фиксирован. Переменная CONMIN - нижняя граница конвейера. Отсюда берутся координаты очередной об-

служиваемой точки. Эта граница - плавающая. Переменная CONMAX - верхняя граница конвейера. Сюда заносятся координаты точек, соседних с обслуживаемой в данный момент, если они не включены. Эта граница также плавающая (растет вверх). На верхнюю границу буфера указывает адрес XXXX. Его Вы можете задать сами в строках 57779 и 57870. В конкретном примере, рассмотренном ниже, этот адрес уже задан в строках DATA (мы были обязаны задать хоть что-то, иначе нельзя было бы получить контрольную сумму) из расчета, что на буфер расходуется 1000 байтов, начиная с адреса 57900. В них можно разместить координаты 500 пикселов на экране, что конечно же немного, но программа работает так, что когда верхняя граница конвейера "дорастает" до физической верхней границы буфера, она вновь устанавливается на нижнюю границу. Это сделать можно, поскольку к этому времени нижняя часть буфера уже очищена и нижняя граница конвейера ушла вверх (это происходит в строке 57879). То же происходит и когда нижняя граница конвейера "дорастает" до верхней границы буфера (57788). Таким образом, наш конвейер работает в отведенном ему ограниченном буфере циклически.

Конец работы программы наступит, когда внутри замкнутого контура не останется ни одного выключенного пиксела. В этом случае на конвейер сверху перестанут поступать координаты и CONMAX перестанет расти, в то же время снизу координаты будут продолжать сниматься и скоро CONMIN достигнет CONMAX, что и явится сигналом конца работы (57793-57803).

Программа работает весьма быстро и довольно элегантно. Кажется, что заполнение идет по всем диагональным направлениям. Верхняя и нижняя границы экрана - непроницаемы для заполнения, зато правая и левая имеют возможность работы "с возвратом". Так, если заполнение дошло до правого края экрана, то оно продолжится слева и наоборот.

```
10 REM *** Загрузчик машинного кода
20 LET adr=57700: LET long=190: LET z=0
30 FOR i=0 TO long-1: READ a
40 POKE (adr+i),a: LET z=z+a
```

```
50 NEXT i
60 LET z=INT (((z/long)-INT (z/long))*long)
70 READ a
80 IF a<>z THEN PRINT "??": STOP

500 REM ***Данные для машинного кода
510 DATA  42,  11,  92,   1,   4
520 DATA   0,   9,  86,  14,   8
530 DATA   9,  94, 237,  83,  44
540 DATA 226, 237,  83,  42, 226
550 DATA  33,  44, 226, 229,  35
560 DATA  35,  34,  40, 226, 225
570 DATA  34,  38, 226,  42,  38
580 DATA 226,  94,  35,  86,  21
590 DATA 205, 207, 225,  42,  38
600 DATA 226,  94,  28,  35,  86

610 DATA 205, 207, 225,  42,  38
620 DATA 226,  94,  35,  86,  20
630 DATA 205, 207, 225,  42,  38
640 DATA 226,  94,  29,  35,  86
650 DATA 205, 207, 225,  42,  38
660 DATA 226,  35,  35, 229,   1
670 DATA  76, 229, 167, 237,  66
680 DATA  32,   5, 225,  33,  44
690 DATA 226, 229, 225,  34,  38
700 DATA 226, 237,  75,  40, 226

710 DATA 167, 237,  66, 200, 195
720 DATA 133, 225, 237,  83,  42
730 DATA 226,  62, 175, 147, 216
740 DATA  95, 167,  31,  55,  31
750 DATA 167,  31, 171, 230, 248
760 DATA 171, 103, 122,   7,   7
770 DATA   7, 171, 230, 199, 171
780 DATA   7,   7, 111, 122, 230
790 DATA   7,  71,   4,  62, 254
800 DATA  15,  16, 253,   6, 255
```

```
          810 DATA 168,  71, 126, 160, 192
          820 DATA 126, 176, 119,  42,  40
          830 DATA 226, 237,  91,  42, 226
          840 DATA 115,  35, 114,  35, 229
          850 DATA   1,  76, 229, 167, 237
          860 DATA  66,  32,   5, 225,  33
          870 DATA  44, 226, 229, 225,  34
          880 DATA  40, 226, 201, 193, 195
          890 DATA  57,   0,   0,   0,   0
```

Дисассемблер программы:

```
57700   2A0B5C         LD HL,(5C0BH)  ;См. с. 109...111
57703   010400         LD BC,0004     ;Сдвиг от DEFADD на 4 бай-
57706       09         ADD HL,BC      ;та (см. с.109...111).
57707       56         LD D,(HL)      ;Координата x.
57708     0E08         LD C,08        ;Сдвиг на
57710       09         ADD HL,BC      ;восемь байтов.
57711       5E         LD E,(HL)      ;Координата y.
57712 ED532CE2         LD(BUFFER),DE  ;Запомнили x,y.
57716 ED532AE2         LD(TEMPXY),DE  ;Запомнили x,y.
57720   212CE2         LD HL,E22C     ;Указание на нижнюю гра-
                                      ;ницу буфера.
57723       E5         PUSH HL        ;Запомнили ее на стеке.
57724       23         INC HL         ;Инициализация верхней
57725       23         INC HL         ;границы
57726   2228E2         LD(CONMAX),HL  ;конвейера.
57729       E1         POP HL         ;Восстановили нижнюю гра-
57730   2226E2         LD(CONMIN),HL  ;ницу буфера и выставили в
                                      ;ней нижнюю границу кон-
                                      ;вейера.
57733   2A26E2 MAIN_L  LD HL,(CONMIN) ;Адрес-указатель на теку-
                                      ;щую координату.
57736       5E         LD E,(HL)      ;Приняли y.
57737       23         INC HL         ;Переход к x.
57738       56         LD D,(HL)      ;Приняли x.
57739       15         DEC D          ;x-1.
57740   CDCFE1         CALL PLOT      ;Проверяем точку слева
                                      ;(x-1),y.
```

```
57743   2A26E2      LD HL,(CONMIN) ;Адрес-указатель на теку-
                                   ;щую координату.
57746       5E      LD E,(HL)      ;Приняли y.
57747       1C      INC E          ;y+1.
57748       23      INC HL         ;Переход к x.
57749       56      LD D,(HL)      ;Приняли x.
57752   CDCFE1      CALL PLOT      ;Проверяем точку вверху
                                   ;x,(y+1).

57755   2A26E2      LD HL,(CONMIN) ;Адрес-указатель на теку-
                                   ;щую координату.
57756       5E      LD E,(HL)      ;Приняли y.
57757       23      INC HL         ;Переход к x.
57758       56      LD D,(HL)      ;Приняли x.
57759       14      INC D          ;x+1.
57760   CDCFE1      CALL PLOT      ;Проверяем точку справа
                                   ;x+1,y.

57763   2A26E2      LD HL,(CONMIN) ;Адрес-указатель на теку-
                                   ;щую координату.
57766       5E      LD E,(HL)      ;Приняли y.
57767       1D      DEC E          ;y-1
57768       23      INC HL         ;Переход к x.
57769       56      LD D,(HL)      ;Приняли x.
57772   CDCFE1      CALL PLOT      ;Проверяем точку внизу
                                   ;x,y-1.
57775   2A26E2      LD HL,(CONMIN) ;Адрес-указатель на теку-
                                   ;щую координату.
57776       23      INC HL         ;Переход к следующей пози-
57777       23      INC HL         ;ции буфера.
57778       E5      PUSH HL        ;Запомнили ее на стеке.
57779   01????      LD BC XXXX     ;XXXX - предельный адрес,
                                   ;до которого может разви-
                                   ;ваться буфер - Вы задаете
                                   ;его сами.
57782       A7      AND A          ;Сброс флагов регистра F.
57783     ED42      SBC HL,BC      ;Проверка на переполнение
                                   ;буфера.
```

```
57785     2005         JR NZ PASS      ;Обход, если буфер еще не
                                       ;переполняется.
57787       E1         POP HL          ;Очистка стека.
57788   212CE2         LD HL,BUFFER    ;Нижняя граница конвейера
                                       ;выставляется в нижнюю
                                       ;границу буфера.
57791       E5         PUSH HL         ;Запомнили ее на стеке.
57792       E1 PASS    POP HL          ;Нижняя граница конвейера.
57793   2226E2         LD(CONMIN),HL   ;Запомнили ее в переменной
57796 ED4B28E2         LD BC,(CONMAX)  ;Вершина конвейера.
57800       A7         AND A           ;Сброс флагов.
57801     ED42         SBC HL,BC       ;Проверка не достигла ли
                                       ;нижняя граница конвейера
                                       ;верхнюю.
57803       C8         RET Z           ;Если да, то конец работы.

57804   C385E1         JP MAIN_L       ;Возврат в вершину глав-
                                       ;ного цикла.
```

Подпрограмма PLOT выполняет четыре задачи. Во-первых, по координатам точки, выставленным в регистровой паре DE определяет адрес в дисплейном файле, соответствующий этой точке (57807 - 57837). Эти операции делаются совершенно так же, как и в процедурах FN f() и FN g() и мы их уже разбирали.

Вторая задача - проверка не включен ли уже пиксел, имеющий данную координату и если да, то возврат (57852 - 57854).

Третья задача - если пиксел не включен, то его надо включить (57855 - 57857).

Четвертая задача - включив пиксел, запомнить его координаты на вершине конвейера и переместить указатель CONMAX на следующую позицию.

```
57807 ED532AE2 PLOT   LD (TEMPXY),DE ;Текущие координаты x,y
57811     3EAF        LD A,0AFH       ;AFH=175 DEC
```

```
57813       93          SUB E           ;Проверка на выход за пре-
57814       D8          RET C           ;делы экрана по вертикали.
57815       5F          LD E,A          ;Дополнение y до 175
57816       A7          AND A           ;Сброс флага переноса.
57817       1F          RRA             ;Ротация вправо.
57818       37          SCF             ;Установка флага переноса.
57819       1F          RRA             ;Ротация вправо.
57820       37          SCF             ;Установка флага переноса.
57821       1F          RRA             ;Ротация вправо.
57822       AB          XOR E           ;Комплексная
57823     E6F8          AND 0F8H        ;операция замещения
57825       AB          XOR E           ;битов 0, 1 и 2.
57826       67          LD H,A          ;Сформировали старший байт
                                        ;адреса в дисплейном файле
57827       7A          LD A,D          ;Координата x.
57828       07          RLCA            ;Вращение влево.
57829       07          RLCA            ;Вращение влево.
57830       07          RLCA            ;Вращение влево.
57831       AB          XOR A           ;Операция замещения
57832     E6C7          AND C7          ;для битов 3,4,5.
57834       AB          XOR A
57835       07          RLCA            ;Вращение влево.
57836       07          RLCA            ;Вращение влево.
57837       6F          LD L,A          ;Сформировали младший байт
                                        ;адреса в дисплейном файле
57838       7A          LD A,D          ;Координата x.
57839     E607          AND 07          ;Маскирование.
57841       47          LD B,A          ;Инициализация счетчика
57842       04          INC B           ;оборотов в регистре B.
57843     3EFE          LD A,0FEH       ;Байт FE=254 интересен
                                        ;тем, что все биты, кро-
                                        ;ме одного, равны едини-
                                        ;це. Вот эту "дырку" мы
                                        ;и вращаем, пока она не
                                        ;встанет на свое место.
57845     0F AGAIN      RRCA            ;Вращение влево.
57846     10FD          DJNZ AGAIN      ;Повтор вращения.
57848     06FF          LD B,0FFH       ;Инверсия, чтобы печать
```

```
57850       A8          XOR B           ;точки была черным по бе-
                                        ;лому, а не наоборот.
57851       47          LD B,A          ;Запомнили в B
57852       7E          LD A,(HL)       ;Приняли в аккумулятор то,
                                        ;что уже содержится на
                                        ;экране в нужной линии.
57853       A0          AND B           ;Логическое сравнение.
57854       C0          RET NZ          ;Если пиксел в текущей
                                        ;координате уже включен,
                                        ;то возврат в вызывающую
                                        ;программу.
57855       7E          LD A,(HL)       ;Приняли в аккумулятор то,
                                        ;что уже содержится на
                                        ;экране в нужной линии.
57856       B0          OR B            ;Включаем требуемый бит.
57857       77          LD (HL),A       ;Включаем требуемый пиксел
57858   2A28E2          LD HL,(CONMAX)  ;Указатель вершины конв-ра
57861 ED5B2AE2          LD DE,(TEMPXY)  ;Текущие x и y.
57865       73          LD (HL),E       ;x
57866       23          INC HL          ;Переход к y.
57867       72          LD (HL),D       ;y
57868       23          INC HL          ;Переместили CONMAX
57869       E5          PUSH HL         ;и запомнили его.
57870   01????          LD BC,XXXX      ;Верхняя граница буфера.
57873       A7          AND A           ;Сброс флагов регистра F.
57874     ED42          SBC HL,BC       ;Проверка заполненности
                                        ;буфера.
57876     2005          JR NZ,PASS_1    ;Обход, если буфер не
                                        ;заполнен.
57878       E1          POP HL          ;Очистка стека
57879   212CE2          LD HL,E22C      ;Если достигнут верхний
                                        ;предел буфера, то верх-
                                        ;нюю границу конвейера
                                        ;выставляем в нижнюю гра-
                                        ;ницу буфера.
57882       E5          PUSH HL         ;И запоминаем ее.
57883       E1 PASS_1   POP HL          ;Верхняя граница конв-ра.
57884   2228E2          LD (CONMAX),HL  ;Запомнили ее.
```

```
57887       C9        RET             ;Возврат.

57894          CONMIN DEFW            ;Нижняя граница конвей-
                                       ;ера.
57896          CONMAX DEFW            ;Верхняя граница конвей-
                                       ;ера.
57898          TEMPXY DEFW            ;Переменная для временного
                                       ;хранения координат теку-
                                       ;щей точки при работе про-
                                       ;цедуры PLOT.
57900          BUFFER DEFW            ;Нижняя граница буфера.
```

Примеры использования процедуры.

Вы можете без труда изобразить на БЕЙСИКе любой замкнутый контур и, задав координаты x,y, принадлежащие области находящейся внутри данного контура произвести его закрашивание цветом INK. Эффект от работы этой процедуры очень напоминает динамическую графику, что способно оживить любую программу, особенно если есть дизайнерские способности и фантазия. если же фантазии пока нет, то посмотрите, как может быть организовано использование этой процедуры на следующих примерах.

Пример 1. Требует подгрузки процедуры рисования отрезков прямых FN g(x,y,p,q).

```
100  DEF FN g(x,y,p,q)=USR 60700
110  DEF FN j(x,y) = USR 57700
120  BORDER 1: PAPER 6: INK 2
130  CLS
140  FOR i=1 TO 12
150  LET x1=i*20
160  LET y1=174
170  LET x2=10+i*20
180  LET y2=20
190  RANDOMIZE FN g(x1,y1,x2,y2)
```

```
200  NEXT i
210  FOR i=1 TO 12
220  LET x1=i*20
230  LET y1=174
240  LET x2=i*20-10
250  LET y2=20
260  RANDOMIZE FN g(x1,y1,x2,y2)
270  NEXT i
280  RANDOMIZE FN g(10,20,130,2)
290  RANDOMIZE FN g(250,20,130,2)
300  PAUSE 100
310  RANDOMIZE FN j(10,5)
```

Пример 2. Требует подгрузки процедуры для изображения прямоугольников FN h(x,y,h,v).

```
100  DEF FN h(x,y,h,v)=USR 60400
110  DEF FN j(x,y) = USR 57700
120  BORDER 2: PAPER 6: INK 2
130  CLS
140  LET x=140
150  FOR j=110 TO 110 STEP -5
160  RANDOMIZE FN h(x,j,60,60)
170  IF x/10 = INT (x/10) THEN RANDOMIZE FN j(x+1,j+1)
180  LET x=x-5
190  NEXT j
```

3.12. Наложение изображений.

Давайте рассмотрим команду OVER стандартного БЕЙСИКа. Вы никогда не задумывались над тем, что эта команда является фактически логической командой?

В БЕЙСИКе существуют логические команды AND, OR, NOT - вот пожалуй и все. Кроме того, при программировании в машинных кодах Вам доступна еще и команда XOR - "Исключающее ИЛИ". Так вот, команда OVER 1 при печати на экране фактически эквивален-

тна функции XOR. Действительно, давайте рассмотрим, что делает команда A XOR B. В результате ее действия включаются те биты, которые были включены либо в A, либо в B, но если они были включены и в A и в B, то они выключаются (в этом отличие от команды OR).

А что делает команда OVER 1? Практически то же самое, но при печати на экране. Когда Вы накладываете одно изображение на другое, включаются те пикселы, которые были включены в одном или в другом изображении, но если пиксел был включен и там и там, то он выключается. Кстати, это довольно эффективный метод стирания, когда в режиме OVER 1 Вы печатаете некоторое изображение поверх самого себя. Попробуйте для эксперимента:

```
10 PRINT AT 10,12; OVER 1: "SPECTRUM"
20 PAUSE 10:
30 GO TO 10
```

Не правда ли, очень похоже на действие команды FLASH, но не совсем то же самое?

В качестве демонстрации возможности печати изображений с наложением по XOR мы рассмотрим программу для наложения отрезков прямых FN k(x,y,p,q). Здесь x,y - координаты исходной точки, а p,q - координаты конечной точки отрезка (абсолютные). Ограничения на величину x,y,p,q, связанные с размером экрана 256x176 очевидны.

```
10 REM *** Загрузчик машинного кода
20 LET adr=57600: LET long=15: LET z=0
30 FOR i=0 TO long-1: READ a
40 POKE (adr+i),a: LET z=z+a
50 NEXT i
60 LET z=INT (((z/long)-INT (z/long))*long)
70 READ a
80 IF a<>z THEN PRINT "??": STOP
```

```
500 REM ***Данные для машинного кода
510 DATA  62, 168,  50, 223, 237
520 DATA 205,  28, 237,  62, 176
530 DATA  50, 223, 237, 201,   0
540 DATA  13,   0,   0,   0,   0
```

Дисассемблер программы:

```
57600     3EA8        LD A,0A8H      ;A8 - это код операции
                                     ;XOR B.
57602   32DFED        LD (EDDF),A    ;В процедуре рисования
                                     ;отрезков прямых FN g()
                                     ;по адресу 60895 стоит
                                     ;код операции B0 (OR B).
                                     ;Он принудительно включа-
                                     ;ет требуемый пиксел на
                                     ;экране. Заменив его на
                                     ;XOR B мы включаем пиксел,
                                     ;если он был выключен. В
                                     ;противном случае - нао-
                                     ;борот выключаем.
57605   CD1CED        CALL ED1C      ;Вызов измененной прог-
                                     ;раммы изображения отрез-
                                     ;ков.
57608     3EB0        LD A,0B0H      ;0B - код операции OR B.
57610   32DFED        LD (EDDF),A    ;В процедуре FN g() вос-
                                     ;становили то, что там и
                                     ;должно было быть.
57613       C9        RET            ;Выход из процедуры.
```

Эта маленькая и очень простая процедура показывает нам пример отнюдь не простой операции. Мы только что сделали изменение машинного кода одной процедуры (FN g()) из другой. Нам и раньше приходилось использовать несколько процедур совместно и при этом они обменивались данными через выделенную для этой цели ячейку памяти. Здесь же имеет место прямое изменение рабочего кода одной процедуры из другой. Это уже совсем другой стиль программирования.

В учебных целях надо сделать еще одно дополнение. Мы хорошо знаем, что находится в процедуре FN g() по адресу 60895 и потому перед выходом из подпрограммы восстановили там первоначальное число, равное B0H. Но не всегда мы заранее можем точно знать, что там было. Поэтому в общем случае положено принять содержимое модифицируемой ячейки памяти, потом сохранить его в какой-либо переменной или на стеке (что менее надежно) и только после этого вносить свои изменения. А перед возвратом следует восстановить то, что было нами нарушено.

Примеры использования процедуры.

Два фантастических узора показывают как простыми графическими средствами можно добиться впечатляющих результатов.

Пример 1. Требует подгрузки процедуры рисования отрезков прямых FN g(x,y,p,q).

```
100  DEF FN k(x,y,p,q)=USR 57600
110  BORDER 0: PAPER 0: INK 3
120  CLS
130  LET s=1
140  LET a=0
150  LET ad=s*PI/128
160  LET x1=80
180  LET y1=88
190  FOR c=1 TO 2
200  FOR i=0 TO 255 STEP s
210  LET x=x1+INT(70*SIN a)
220  LET y=y1+INT(70*COS a)
230  RANDOMIZE FN k(x,y,x1,y1)
240  LET a=a+ad
250  NEXT i
260  LET x1=x1+96
270  NEXT c
280  PAUSE 100
290  GO TO 110
```

Пример 2. Требует подгрузки процедуры рисования отрезков прямых FN g(x,y,p,q).

```
100  DEF FN k(x,y,p,q)=USR 57600
110  BORDER 5: PAPER 5: INK 1
120  CLS
130  LET s=1
140  LET a=0
150  LET ad=s*PI/128
160  LET x1=127
180  LET y1=88
190  FOR i=0 TO 255 STEP s
200  LET x=x1+INT(110*SIN a)
200  LET y=y1+INT(70*COS a)
210  RANDOMIZE FN k(x,y,x1,y1)
220  LET a=a+ad
230  NEXT i
240  PAUSE 0
250  GO TO 110
```

3.13. Увеличение изображений.

Если Вы работали с графическим редактором ARTSTUDIO, то не могли не восхититься изяществом того, как происходит увеличение экрана в 2 раза (MAGNIFY X 2). Теперь Вы можете это сделать и сами. Достигающийся при этом эффект способен оживить очень многие программы.

Процедура позволяет увеличить в 2 раза как по горизонтали, так и по вертикали все, что находится в заданном Вами "окне". При этом координаты и размеры выбранного Вами "окна" задаются в знакоместах, а не в абсолютных координатах. Прежде чем увеличивать Ваше изображение, процедура сделает его копию и сохранит ее в верхних областях памяти (адрес временного буфера можно задать самим) и только после этого начнет перестраивать экран. Эту особенность процедуры можно и нужно использовать например для того, чтобы в любой момент можно было бы подать команду и вернуться к исходному изображению.

Вместе с тем, понятно, что для процедуры совершенно все равно, что ей надо увеличивать, поэтому увеличив изображение в два раза и получив новый экран, Вы можете подать команду еще раз и вновь увеличить экран в два раза. Так можно увеличивать изображение во много раз. Правда, буфер хранящий исходное изображение,- только один и потому возвратиться Вы можете ТОЛЬКО К ПРЕДЫДУЩЕМУ изображению. Так, если Вы выполните увеличение дважды, то есть увеличите рисунок в 4 раза, то возврат вернет Вас к изображению, увеличенному в 2 раза, а исходное будет уже утрачено.

Назовем эту процедуру FN l(x,y,h,v), где:

x - горизонтальная координата левого верхнего угла "окна" (задается в знакоместах, - x < 32);
y - вертикальная координата левого верхнего угла "окна" (задается в знакоместах, - y < 22);
h - размер "окна" по горизонтали (ширина "окна" - задается в знакоместах, - 2h + x < 32);
v - размер "окна" по вертикали (высота "окна" - задается в знакоместах, - 2v + y < 32);

Процедура задается командой DEF FN l(x,y,h,v) = USR 56700 и вызывается командой RANDOMIZE FN l(x,y,h,v).

Для восстановления исходного изображения, содержащегося в буфере можно воспользоваться точкой входа в подпрограмму COPY_D, которая находится по адресу 56957. Задайте функцию без параметров, например DEF FN m() = USR 56957 и когда хотите восстановить на экране исходное изображение пользуйтесь командой RANDOMIZE FN m().

```
10 REM *** Загрузчик машинного кода
20 LET adr=56700: LET long=285: LET z=0
30 FOR i=0 TO long-1: READ a
40 POKE (adr+i),a: LET z=z+a
50 NEXT i
60 LET z=INT (((z/long)-INT (z/long))*long)
```

```
70 READ a
80 IF a<>z THEN PRINT "??": STOP

500 REM ***Данные для машинного кода
510 DATA  42,  11,  92,   1,   4
520 DATA   0,   9,  86,  14,   8
530 DATA   9,  94, 237,  83, 137
540 DATA 222,   9, 126,  50, 140
550 DATA 222,   9, 126,  50, 139
560 DATA 222,  58, 138, 222,  71
570 DATA  58, 140, 222, 128, 230
580 DATA 224,  40,   6,  62,  31
590 DATA 144,  50, 140, 222,  58
600 DATA 137, 222,  71,  58, 139

610 DATA 222, 128, 214,  22,  56
620 DATA   6,  62,  21, 144,  50
630 DATA 139, 222, 237,  91, 137
640 DATA 222, 123, 230,  24, 246
650 DATA  64, 103, 123, 230,   7
660 DATA 183,  31,  31,  31,  31
670 DATA 130, 111,  34, 141, 222
680 DATA  17,   0,  64, 167, 237
690 DATA  82,  17,   0, 118,  25
700 DATA  34, 143, 222, 205, 113

710 DATA 222,  42, 141, 222, 237
720 DATA  91, 143, 222,  58, 139
730 DATA 222,  71, 197,   1,   2
740 DATA   4, 197, 205,  20, 222
750 DATA 193,  16, 249,  42, 141
760 DATA 222, 205,  79, 222,  34
770 DATA 141, 222,   6,   4,  13
780 DATA  32, 235, 237,  91, 143
790 DATA 222, 205,  89, 222, 237
800 DATA  83, 143, 222, 193,  16

810 DATA 217, 201,  58, 140, 222
```

```
 820 DATA  71,  34, 145, 222, 237
 830 DATA  83, 147, 222, 197, 205
 840 DATA  54, 222, 193,  16, 249
 850 DATA  42, 145, 222, 237,  91
 860 DATA 147, 222, 229, 205,  99
 870 DATA 222, 225,  36,  36,  20
 880 DATA 201,  26,   1,   2,   4
 890 DATA 197, 245, 175, 119, 241
 900 DATA  23, 245, 203,  22, 241

 910 DATA 203,  22,  16, 247,  35
 920 DATA 193,  13,  32, 237,  19
 930 DATA 201,  62,  32, 133, 111
 940 DATA 208,  62,   8, 132, 103
 950 DATA 201,  62,  32, 131,  95
 960 DATA 208,  62,   8, 130,  87
 970 DATA 201,  58, 140, 222, 203
 980 DATA  39,  71, 126,  36, 119
 990 DATA  37,  35,  16, 249, 201
1000 DATA  33,   0,  64,  17,   0

1010 DATA 118,   1,   0,  26, 237
1020 DATA 176, 201,  33,   0, 118
1030 DATA  17,   0,  64,   1,   0
1040 DATA  26, 237, 176, 201,   2
1050 DATA   2,   5,  10, 130,  72
1060 DATA 226, 118,  98,  78, 194
1070 DATA 125,   0,   0,   0,   0
1080 DATA  38,   0,   0,   0,   0
```

Дисассемблер программы:

1. На первом этапе процедура принимает параметры и сохраняет их в соответствующих ячейках памяти.

```
56700   2A0B5C        LD HL,(5C0BH)  ;См. с.109...111
56703   010400        LD BC,0004     ;Сдвиг от DEFADD на 4 бай-
56706       09        ADD HL,BC      ;та (см. с.109...111)
```

```
56707       56          LD D,(HL)       ;Координата x.
56708     0E08          LD C,08         ;Сдвиг на
56710       09          ADD HL,BC       ;восемь байтов.
56711       5E          LD E,(HL)       ;Координата y.
56712 ED5389DE          LD(COORDY),DE   ;Запомнили x,y.
56716       09          ADD HL,BC       ;Сдвиг на восемь байтов.
56717       7E          LD A,(HL)       ;Параметр h.
56718   328CDE          LD (WIDTH),A    ;Запомнили его.
56721       09          ADD HL,BC       ;Сдвиг на восемь байтов.
56722       7E          LD A,(HL)       ;Параметр v.
56723   328BDE          LD (HEIGHT),A   ;Запомнили его.
```

2. Второй этап - первичные проверки и настройки.

```
56726   3A8ADE          LD A,(COORDX)   ;Координата x.
56729       47          LD B,A          ;Координата x.
56730   3A8CDE          LD A,(WIDTH)    ;Ширина окна h.
56733       80          ADD A,B         ; x+h
56734     E6E0          AND E0          ;E0H=1110 0000 BIN
                                        ;Результатом этой операции
                                        ;может быть 0 только в том
                                        ;случае, если в аккумуля-
                                        ;торе выключены три стар-
                                        ;ших бита, то есть h<=31
56736     2806          JR Z,PASS_1     ;В этом случае все O.K. и
                                        ;делаем обход на PASS_1.
56738     3E1F          LD A,1FH        ;31
56740       90          SUB B           ;31-x
56741   328CDE          LD (WIDTH),A    ;hmax=31-x - максимально
                                        ;допустимое значение h.
56744   3A89DE PASS_1   LD A,(COORDY)   ;y
56747       47          LD B,A          ;y
56748   3A8BDE          LD A,(HEIGHT)   ;v
56751       80          ADD A,B         ;y+v
56752     D616          SUB 16H         ;Проверка на <22
56754     3806          JR C,PASS_2     ;Если y меньше 22, то все
                                        ;O.K. и делаем обход.
56756     3E15          LD A,15H        ;21
```

```
56758       90          SUB B           ;21-y
56759   328BDE          LD (HEIGHT),A   ;vmax=21-y - вводим макси-
                                        ;мально допустимое значе-
                                        ;ние v.
```

3. На третьем этапе по координатам x и y (заданы в знакоместах) определяем адрес в дисплейном файле, соответствующий левому верхнему углу выделенного нами "окна" и помещаем его в регистровую пару HL и в программную переменную ADDR.

```
56762 ED5B89DE PASS_2 LD DE,(COORDY) ;x,y
56766       7B        LD A,E         ;Координата y     000?????
56767     E618        AND 18H        ;Выделение сег-
                                     ;мента экрана     000??000
56769     F640        OR 40          ;Указание на
                                     ;дисплейный файл  010??000
56771       67        LD H,A         ;Выставили H (см. рис.   )
56772       7B        LD A,E         ;Координата y     000?????
56773     E607        AND 07         ;Выделили номер
                                     ;ряда в сегменте  00000???
56775       B7        OR A           ;Очистили флаг "C"
56776       1F        RRA            ;Вращение         000000??
56777       1F        RRA            ;Вращение         ?000000?
56778       1F        RRA            ;Вращение         ??000000
56779       1F        RRA            ;Вращение         ???00000
56780       82        ADD A,D        ;Прибавили номер
                                     ;столбца          ????????
56781       6F        LD L,A         ;Младший байт адреса в
                                     ;дисплейном файле.
56782   228DDE        LD (ADDR),HL   ;Запомнили адрес в пере-
                                     ;менной ADDR.
```

4. Зная адрес, с которого начинается наше "окно" в дисплейном файле, мы можем теперь сохранить текущее графическое изображение в верхних областях памяти во временном буфере. В качестве примера здесь принят начальный адрес буфера 7600 H = 30208 DEC. Но Вы можете поменять его после того как программа будет уже набрана (если сделать это раньше, может не сойтись

контрольная сумма). О том, как это сделать, мы укажем после описания процедуры.

Таким образом, в буфере получается копия исходного изображения.

```
56785   110040        LD DE 4000      ;Начало дисплейного файла.
56788       A7        AND A           ;Очистка флага переноса.
56789     ED52        SBC HL,DE       ;Определили "смещение" ад-
                                      ;реса начала окна относи-
                                      ;тельно начала дисплейного
                                      ;файла.
56791   110076        LD DE 7600      ;Начало буфера.
56794       19        ADD HL,DE       ;Прибавив к нему смещение,
                                      ;получим начальный адрес
                                      ;нашего "окна" в копии.
56795   228FDE        LD (ADDR_1),HL  ;Запомнили этот адрес.
56798   CD71DE        CALL COPYUP     ;Переброска копии вверх.
```

5. Теперь, когда у нас есть копия исходного изображения в буфере, мы можем начать строить удвоенное изображение на экране, перебрасывая байт за байтом из буфера на экран и производя при этом удвоение его образа как по горизонтали, так и по вертикали. Логика работы при этом очень похожа на логику программ FN (d) и FN e(), которые печатали текст символами двойного размера. Фактическая разница в том, что там образ брался из генератора шрифта ПЗУ, а здесь образ берется из буфера.

Сначала организуется цикл по вертикали (по рядам). В результате вместо одного ряда шириной h исходного изображения мы будем иметь 2 ряда шириной 2h на экране.

```
56801   2A8DDE        LD HL,(ADDR)    ;адрес в дисп. файле
56804 ED5B8FDE        LD DE,(ADDR_1)  ;адрес в копии
56808   3A8BDE        LD A,(HEIGHT)   ;v.
56811       47        LD B,A          ;Подготавливаем цикл по
                                      ;горизонтальным рядам.
                                      ;Количество рядов = v.
56812       C5 LOOP_1 PUSH BC         ;Вершина цикла по рядам.
```

```
56813   010204        LD BC 0402      ;Подготавливаем еще 2 вло-
                                      ;женных цикла. Внешний -
                                      ;на 2 прохода (C) и внут-
                                      ;ренний на 4 прохода (B).
56816       C5 LOOP_2 PUSH BC         ;Вершина внутреннего и
                                      ;внешнего вложенных циклов
```

Цикл по горизонтали (по столбцам) для данного ряда организуется в процедуре DOUBLE.

```
56817   CD14DE        CALL DOUBLE     ;8 раз вызывается проце-
                                      ;дура DOUBLE.
56820       C1        POP BC
56821     10F9        DJNZ LOOP_2     ;Конец внутреннего цикла
                                      ;по B (4 прохода).
56823   2A8DDE        LD HL,(ADDR)    ;Адрес в дисп.файле.
56826   CD4FDE        CALL NEW_R      ;Переход к новому ряду.
56829   228DDE        LD (ADDR),HL    ;Запомнили новый адрес
                                      ;в дисплейном файле.
56832     0604        LD B,04
56834       0D        DEC C           ;
56835     20EB        JR NZ,LOOP_2    ;Конец внешнего цикла
                                      ;по C (2 прохода).
56837 ED5B8FDE        LD DE,(ADDR_1)  ;Установив новый адрес
56841   CD59DE        CALL NEW_R_1    ;в дисплейном файле, мы
56844 ED538FDE        LD (ADDR_1),DE  ;переходим и к новому
                                      ;адресу в буфере.
56848       C1        POP BC
56849     10D9        DJNZ LOOP_1     ;Конец цикла по рядам.
56851       C9        RET
```

Процедура DOUBLE организует цикл по горизонтали для данного экранного ряда. Она вызывает процедуру DOUB_L, которая выполняет цикл по восьми линиям данного знакоместа.

```
56852   3A8CDE DOUBLE LD A,(WIDTH)    ;Ширина "окна"
56855       47        LD B,A          ;Ширина "окна"
56856   2291DE        LD (ADDR_2),HL  ;Временно запомнили адрес
                                      ;в дисплейном файле.
```

```
56859 ED5393DE        LD (ADDR_3),DE ;Временно запомнили адрес
                                     ;в копии дисплейного файла
56863       C5 LOOP_3 PUSH BC        ;Вершина цикла по ширине
                                     ;"окна".
56864   CD36DE        CALL DOUB_L    ;Удвоение линии по гори-
                                     ;зонтали.
56867       C1        POP BC
56868     10F9        DJNZ LOOP_3    ;Конец цикла по ширине
                                     ;"окна".
56870   2A91DE        LD HL,(ADDR_2) ;Перед выходом восстано-
56873 ED5B93DE        LD DE,(ADDR_3) ;вили испорченные при ра-
                                     ;боте процедуры значения
                                     ;в HL и DE.
56877       E5        PUSH HL
```

Мы удвоили линию по горизонтали, растянув ее в два раза по ширине экрана. Каждому пикселу исходной линии (из буфера) соответствует пара пикселов в новой линии (на экране). Теперь, чтобы удвоить ее и по вертикали, повторяем ее еще раз на один пиксел ниже. Этим занимается процедура REPEAT.

```
56878   CD63DE        CALL REPEAT    ;Повторение линии.
56881       E1        POP HL
56882       24        INC H
56883       24        INC H
56884       14        INC D
56885       C9        RET
```

Процедура DOUB_L выполняет удвоение одной линии в одном знакоместе. Побитная раскладка линии берется сверху (DE) и с удвоением опускается в дисплейный файл (HL).

Ее логику работы мы разобрали на стр. 132 и здесь не будем на ней останавливаться.

```
56886       1A DOUB_L LD A,(DE)
56887   010204        LD BC,0402
56890       C5 LOOP_4 PUSH BC
```

```
56891       F5         PUSH AF
56892       AF         XOR A
56893       77         LD (HL),A
56894       F1         POP AF
56895       17 LOOP_5  RLA
56896       F5         PUSH AF
56897     CB16         RL (HL)
56899       F1         POP AF
56900     CB16         RL (HL)
56902     10F7         DJNZ LOOP_5
56904       23         INC HL
56905       C1         POP BC
56906       0D         DEC C
56907     20ED         JR NZ,LOOP_4
56909       13         INC DE
56910       C9         RET
```

По адресу начала экранного ряда, содержащегося в HL, процедура NEW_R определяет адрес начала следующего нижележащего ряда. При этом учитывается, что следующий ряд может принадлежать другому экранному сегменту.

```
56911     3E20 NEW_R   LD A,20H       ;Переход к новому ряду
56913       85         ADD A,L        ;Проверка на переполнение
56914       6F         LD L,A         ;экранного сегмента.
56915       D0         RET NC         ;Если все О.К., то возврат
56916     3E08         LD A,08        ;В противном случае изме-
                                      ;няется значение в регист-
56918       84         ADD A,H        ;ре H, т.е. выполняется
56919       67         LD H,A         ;переход к новому сегменту
56920       C9         RET            ;Возврат в вызывающую про-
                                      ;цедуру.
```

Как процедура NEW_R отыскивала адрес начала очередного ряда в дисплейном файле, точно так же процедура NEW_R_1 отыскивает адрес начала нового ряда в копии исходного файла. т.е. в буфере.

```
56921     3E20 NEW_R_1LD A,20H       ;Переход к новому ряду
56923       83         ADD A,E       ;Проверка на переполнение
56924       5F         LD E,A        ;экранного сегмента.
56925       D0         RET NC        ;Если все О.К., то возврат
56926     3E08         LD A,08       ;В противном случае изме-
                                     ;няется значение в регист-
56928       82         ADD A,D       ;ре H, т.е. выполняется
56929       57         LD D,A        ;переход к новому сегменту
56930       C9         RET           ;Возврат в вызывающую про-
                                     ;цедуру.
```

Процедура REPEAT повторяет на экране текущую линию на один пиксел ниже.

```
56931   3A8CDE REPEAT LD A,(WIDTH)   ;Ширина "окна".
56934     CB27        SLA A          ;Удвоенная ширина
56936       47        LD B,A         ;"окна".
56937       7E LOOP_6 LD A,(HL)      ;Адрес в дисплейном файле.
56938       24        INC H          ;Опустились на одну линию
                                     ;ниже.
56939       77        LD (HL),A      ;Повторили изображение
                                     ;верхней линии.
56940       25        DEC H          ;Вернулись к верхней линии
56941       23        INC HL         ;Перешли к следующему зна-
                                     ;коместу.
56942     10F9        DJNZ LOOP_6    ;Повторяем процесс, пока
                                     ;пройдем всю ширину окна.
56944       C9        RET            ;Возврат в вызывающую
                                     ;процедуру.
```

Процедура COPYUP копирует содержимое дисплейного файла в отведенный буфер.

```
56945   210040 COPYUP LD HL,4000H    ;Адрес источника - начало
                                     ;дисплейного файла.
56948   110076        LD DE,7600H    ;Адрес места назначения -
                                     ;30208 (можно изменить).
```

```
56951   01001A          LD BC,1A00H     ;Количество перебрасывемых
                                        ;байтов.
56954     EDB0          LDIR            ;Команда на переброску.
56956       C9          RET             ;Выход.
```

Процедура COPY_D восстанавливает первоначальное изображение из буфера на экран путем копирования вниз.

```
56957   210076 COPY_D   LD HL,7600H     ;Адрес источника - начало
                                        ;буфера (можно изменить).
56960   110040          LD DE,4000H     ;Адрес места назначения -
                                        ;начало дисплейного файла.
56963   01001A          LD BC,1A00H     ;Количество перебрасывемых
                                        ;байтов.
56966     EDB0          LDIR            ;Команда на переброску.
56968       C9          RET             ;Выход.

56969           COORDY  DEFB            ;Вертикальная координата
                                        ;левого верхнего угла
                                        ;"окна", подлежащего
                                        ;увеличению.
56970           COORDX  DEFB            ;Горизонтальная координата
                                        ;левого верхнего угла
                                        ;исходного "окна".
56971           HEIGHT  DEFB            ;Высота "окна" (в знако-
                                        ;местах.
56972           WIDTH   DEFB            ;Ширина "окна" (в знако-
                                        ;местах).
56973           ADDR    DEFW            ;Адрес в дисплейном фай-
                                        ;ле, соответствующий
                                        ;текущей координате печати
56975           ADDR_1  DEFW            ;Адрес в буферном фай-
                                        ;ле, соответствующий
                                        ;текущей координате печати
56977           ADDR_2  DEFW            ;То же, что и ADDR, но для
                                        ;временного хранения во
                                        ;время работы процедуры
                                        ;DOUBLE.
```

```
56979          ADDR_3 DEFW          ;То же, что и ADDR_1, но
                                    ;для временного хранения
                                    ;во время работы
                                    ;процедуры DOUBLE.
```

В заключение описания этой процедуры мы должны сказать несколько слов о буфере, в котором хранится исходное изображение. Здесь предполагается, что он начинается с адреса 30208. На это значение настроен машинный код, приведенный в загручике процедуры в строках DATA, на него же настроена проверка контрольной суммы в этом загрузчике. После того, как Вы введете процедуру в память, ничто уже не может помешать Вам изменить этот адрес так, как Вам удобно и выгрузить перенастроенную процедуру на ленту. В процессе работы программы Вы можете динамически из БЕЙСИКа менять начальный адрес этого буфера операторами POKE, обращая внимание на то, чтобы этот буфер не затер содержащийся в памяти где-либо машинный код или Вашу БЕЙСИК-программу.

Изменения надо внести в ячейки:

```
56792,56793  - в главной процедуре;
56949,56950  - в процедуре COPYUP;
56958,56959  - в процедуре COPY_D.
```

При внесении изменений не забудьте, что в машинном коде хранится сначала младший байт адреса и только потом старший. Так, если Вам надо разместить буфер начиная с адреса 42005, то Вы можете найти старший байт:

```
HB = INT (42005/256) = 164
```

и младший байт:

```
LB = 42005 - HB*256 = 42005 - 164*256 = 21
```

Изменения в коде выполните оператором POKE:

```
POKE 56792,19    POKE 56793,164      POKE 56949,19
POKE 56950,164   POKE 56958,19       POKE 56959,164
```

Мы не станем приводить примеров работы этой процедуры. Вы с успехом можете сами выполнить любое доступное Вам изображение на экране и, дав команду увеличить его в два раза, а потом вернуться к исходному. Мы только ограничимся следующими рекомендациями:

1. Если увеличиваемое изображение - многоцветное, то Вам не избежать проблем с "клэшингом" атрибутов, ведь процедура FN l(x,y,h,v) растягивает в два раза только пиксельное изображение и не трогает атрибутов. Поэтому желательно, чтобы увеличиваемое изображение было бы монохромным.

2. Имеет смысл установить цвет PAPER увеличенного изображения. Для этого прежде, чем делать увеличение в два раза, целесообразно предварительно окрасить "удвоенное окно" заданным Вами цветом PAPER, для чего можно воспользоваться процедурой FN c(x,y,h,v,c,b,f).

3. Увеличенное изображение будет лучше смотреться, если его выделить из общего поля экрана с помощью прямоугольной рамки, для чего после окрашивания в цвет PAPER можно воспользоваться процедурой изображения прямоугольников FN h(x,y,h,v).

3.14. Компрессия экрана.

Если Ваша программа содержит большое количество графических изображений, причем они не исполняются самой программой, а подготовлены ранее, например в графическом редакторе, то возникает вопрос о необходимости каким-либо образом компрессировать изображения, обеспечив возможности хранения максимума экранов в ограниченном объеме памяти.

Все методы компрессии данных строятся на том, что надо выявить повторяющиеся последовательности и заменить их на код, который содержит указание на то, что это за последовательность

и каков при этом коэффициент повторения. На сегодняшний день известны самые разнообразные приемы, например основанные на том, что наиболее часто повторяющиеся символы, такие как "А" кодируются не кодом ASCII (который равен 65), а своим каким-то кодом, например 01, соответственно символ "Е" например кодом 02 и т.п. В результате получается, что битовая конструкция символов содержит много нулей, которые можно "ужать", т.е. на символ в среднем расходуется не 8 битов, как обычно, а в среднем 4-5 (до 6). Такой метод применим при компрессии текстовых блоков общего назначения.

Если текст имеет специфику, например это текст адвентюрной программы, в которой часто повторяются определенные слова и словосочетания, то компрессия может основываться на создании резидентного словаря на 256 позиций и замене этих слов или словосочетаний по тексту на их порядковый номер. Излишне говорить о том, что и генерацию такого словаря и кодирование текста выполняют не вручную, а написав для этого служебную процедуру.

В нашем случае речь идет о компрессии графики, а она обладает той спецификой, что здесь особенно часто встречаются повторяющиеся символы. Так, например, сплошная горизонтальная линия через весь экран представляет из себя 32 следующих подряд байта, каждый из которых равен 255. Этим обычно и пользуются при компрессии графики. Мы должны отметить, что алгоритмов компрессии графики может быть бесконечно много и ни один из них не является абсолютным. То есть для каждого алгоритма можно подобрать такую раскладку экрана, при которой он будет выполнять максимальное сжатие, но для каждого же можно подобрать и такие условия, при которых компрессированный файл окажется больше исходного по величине. В свое время этот вопрос получил некоторое обсуждение на страницах "ZX-РЕВЮ" (N1, 1991,с. 6, N2, 1991,с. 24).

Процедура компрессии "снимает" изображение с экрана, компрессирует его и отправляет на хранение в выделенную для этого область оперативной памяти компьютера, после чего готова к

приему нового экрана. Компрессированные изображения сохраняются в оперативной памяти последовательно и могут быть восстановлены оттуда с помощью декомпрессирующей программы.

Назовем процедуру компрессии FN n(h,l), где:
h - старший байт адреса, с которого начинается область, отведенная для компрессированных изображений;
l - младший байт этого адреса.
h,l < 255.
Задание процедуры: DEF FN n(h,l) = USR 56600
Вызов процедуры : RANDOMIZE FN n(h,l)

```
10 REM *** Загрузчик машинного кода
20 LET adr=56600: LET long=60: LET z=0
30 FOR i=0 TO long-1: READ a
40 POKE (adr+i),a: LET z=z+a
50 NEXT i
60 LET z=INT (((z/long)-INT (z/long))*long)
70 READ a
80 IF a<>z THEN PRINT "??": STOP

500 REM ***Данные для машинного кода
510 DATA  42,  11,  92,   1,   4
520 DATA   0,   9,  86,  14,   8
530 DATA   9,  94, 237,  83,  82
540 DATA 221,  33,   0,  64,   6
550 DATA   1, 126,  44,  32,   8
560 DATA  36, 245, 124, 254,  91
570 DATA  40,  16, 241,  78, 185
580 DATA  32,   4,   4,  32, 238
590 DATA   5,  18,  19, 120,  18
600 DATA  19,  24, 227, 241,  18

610 DATA  19, 120,  18, 237,  83
620 DATA  22, 221, 201,   0,   0
630 DATA  56,   0,   0,   0,   0
```

Дисассемблер программы:

```
56598          NEXT_S DEFW           ;В эту программную пере-
                                     ;менную заносится адрес, с
                                     ;которого может быть нача-
                                     ;то хранение очередного
                                     ;экрана после того, как
                                     ;данная процедура отрабо-
                                     ;тает.
56600   2A0B5C         LD HL,(5C0BH) ;См. с. 109...111
56603   010400         LD BC,0004    ;Сдвиг от DEFADD на 4 бай-
56606       09         ADD HL,BC     ;та (см. с.109...111).
56607       56         LD D,(HL)     ;Параметр h.
56608     0E08         LD C,08       ;Сдвиг на
56610       09         ADD HL,BC     ;восемь байтов.
56611       5E         LD E,(HL)     ;Параметр l.
56612 ED5352DD         LD(ADDR),DE   ;Запомнили h и l в прог-
                                     ;раммной переменной по ад-
                                     ;ресу 56658.
56616   210040         LD HL,4000    ;Адрес начала дисплейного
                                     ;файла.
56619     0601 RETURN  LD B,01       ;Инициализация счетчика
                                     ;повторяющихся байтов.
56621       7E         LD A,(HL)     ;Приняли текущий байт.
56622       2C AGAIN   INC L         ;И перешли к следующему.
56623     2008         JR NZ,PASS    ;Если регистр L еще не пе-
                                     ;реполнен, то обход.
56625       24         INC H         ;В противном случае нара-
                                     ;щиваем старший регистр H.
56626       F5         PUSH AF       ;Проверка
56627       7C         LD A,H        ;на конец
56628     FE5B         CP 5B         ;экранной области.
56630     2810         JR Z,END      ;Переход на завершение ра-
                                     ;боты, если экран исчерпан
56632       F1         POP AF
56633       4E PASS    LD C,(HL)     ;Очередной байт приняли в
                                     ;регистр C и сравнили его
56634       B9         CP C          ;с предыдущим в A.
```

```
56635     2004         JR NZ,PASS_1    ;Если нет повтора, то об-
                                       ;ход на PASS_1.
56637       04         INC B           ;Если есть повтор, то на-
                                       ;ращиваем счетчик повтора
56638     20EE         JR NZ,AGAIN     ;и переходим на AGAIN для
                                       ;проверки очередного байта
                                       ;экрана.
56640       05         DEC B           ;Если счетчик обнулился,
                                       ;т.е. переполнился, умень-
                                       ;шаем его на единицу и пе-
                                       ;рестаем наращивать.
56641       12 PASS_1  LD (DE),A       ;Перенесли байт из экрана
                                       ;в буфер и указатель в бу-
56642       13         INC DE          ;фере переставили вверх.
56643       78         LD A,B          ;Ввели коэффициент повтора
56644       12         LD (DE),A       ;и заслали его в буфер.
56645       13         INC DE          ;Переставили указатель в
                                       ;буфере вверх.
56646     18E3         JR RETURN       ;Возврат для обработки
                                       ;очередного байта.
56648       F1 END     POP AF          ;Восстановление стека.
56649       12         LD (DE),A       ;Последний байт выдается
                                       ;в буфер
56650       13         INC DE          ;Туда же выдается его
56651       78         LD A,B          ;коэффициент повтора
56652       12         LD (DE),A       ;из регистра B.
56653 ED5316DD         LD (NEXT_S),DE  ;Адрес конца буфера фикси-
                                       ;руется в переменной.
                                       ;Он может быть использован
                                       ;например для того, чтобы
                                       ;знать где можно начинать
                                       ;сохранение следующего эк-
                                       ;рана.
56657       C9         RET
56658          ADDR    DEFW            ;Адрес, с которого начи-
                                       ;нается в ОЗУ область хра-
                                       ;нения нашего скомпресси-
                                       ;рованного экрана.
```

3.15. Декомпрессия экрана.

Файлы скомпрессированных экранов можно восстановить из ОЗУ и снова поместить на экран. Для этого служит декомпрессирующая программа FN o(h,l). Здесь:

h - старший байт адреса, начиная с которого хранится Ваше компрессированное изображение;
l - младший байт этого адреса.
h,l < 255.
Задание процедуры: DEF FN o(h,l) = USR 56500
Вызов процедуры : RANDOMIZE FN o(h,l)

```
10 REM *** Загрузчик машинного кода
20 LET adr=56500: LET long=35: LET z=0
30 FOR i=0 TO long-1: READ a
40 POKE (adr+i),a: LET z=z+a
50 NEXT i
60 LET z=INT (((z/long)-INT (z/long))*long)
70 READ a
80 IF a<>z THEN PRINT "??": STOP

500 REM ***Данные для машинного кода
510 DATA  42,  11,  92,   1,   4
520 DATA   0,   9,  86,  14,   8
530 DATA   9,  94,  33,   0,  64
540 DATA  26, 245,  19,  26,  19
550 DATA  71, 241, 119,  35,  16
560 DATA 252, 124, 254,  91,  32
570 DATA 240, 201,   0,   0,   0
580 DATA  28,   0,   0,   0,   0
```

В результате работы компрессирующей программы сжатое изображение хранится в оперативной памяти и имеет следующий формат:

B1	K1	B2	K2	.....	Bi	Ki	.....

Здесь Bi - i-ый байт изображения, а Ki - коэффициент его повтора (от 0 до 255). Процедура декомпрессии предельно проста. Байты из памяти берутся парами. Коэффициент повтора становится параметром цикла и в этом цикле байт изображения помещается на экран (на адрес в экранной области указывает регистровая пара HL). Сигналом к окончанию работы является момент выхода H за пределы, отведенные для экранной области.

Дисассемблер программы:

```
56500   2A0B5C        LD HL,(5C0BH)  ;См. с. 109...111.
56503   010400        LD BC,0004     ;Сдвиг от DEFADD на 4 бай-
56506      09         ADD HL,BC      ;та (см. с. 109...111).
56507      56         LD D,(HL)      ;Параметр h.
56508     0E08        LD C,08        ;Сдвиг на
56510      09         ADD HL,BC      ;восемь байтов.
56511      5E         LD E,(HL)      ;Параметр l.
56512   210040        LD HL,4000     ;Начало дисплейного файла.
56515      1A AGAIN   LD A,(DE)      ;Приняли байт из оператив-
                                     ;ной памяти.
56516      F5         PUSH AF        ;Запомнили его на стеке.
56517      13         INC DE         ;Переход к новому байту.
56518      1A         LD A,(DE)      ;Приняли очередной байт
                                     ;(это коэффициент повтора)
56519      13         INC DE         ;Переход к новому байту.
56520      47         LD B,A         ;В регистре B организуется
                                     ;счетчик повторов.
56521      F1         POP AF         ;Восстановили байт экрана.
56522      77 LOOP    LD (HL),A      ;И поместили его на экран.
56523      23         INC HL         ;Следующий байт экрана.
56524     10FC        DJNZ LOOP      ;Повторяем цикл от LOOP
                                     ;столько раз, каков коэф-
                                     ;фициент повтора.
56526      7C         LD A,H         ;Проверяем на окончание
56527     FE5B        CP 5BH         ;экранную область памяти.
56529     20F0        JR NZ,AGAIN    ;Если еще не конец, то
                                     ;продолжаем работу.
56531      C9         RET            ;Возврат.
```

Уважаемый читатель!

Скоро исполнится десять лет со дня выхода на широкую арену первого "Спектрума". По нашим данным первую партию компьютеров этой серии сэр К.Синклер продемонстрировал на предновогодней выставке-продаже в декабре 1982 года.

За это время в мире накоплен огромный теоретический и практический опыт работы с этой самой массовой и доступной машиной. Самый большой пласт открытий и достижений находится в тех тысячах высококачественных программ, которые доступны Вам для работы, отдыхы, развлечения и, самое главное - обучения.

Компьютерная программа - не книжка с картинками, в которой все просто и понятно, но это, тем не менее, все-таки учебник для того, кто любит исследовательскую работу. Трудно работать с учебником, в котором все непонятно и здесь мы видим свою главную задачу - дать основные понятия, показать наезженные и освоенные приемы и подходы. Если теперь разбираясь с машинным кодом фирменной программы Вы хоть чуть-чуть почувствуете ощущение, что где-то я это видел и это мне знакомо, значит наша цель наполовину достигнута.

Конечно, мы затронули только самый край всего того, что накоплено в области компьютерной графики. Но, во первых, мы с Вами не прощаемся и готовим новые книги, а во-вторых, мы надеемся, что Вы начнете самостоятельное планомерное исследование.

Пройдут годы, сменятся модели компьютеров и языки программирования, но опыт, который Вы сейчас приобретаете с маленьким "Спектрумом", останется точно так же, как останутся основные алгоритмы и приемы, а самое главное - останется и закрепится особый алгоритмический образ мышления, который поможет Вам легко освоить любую технику и любые языки программирования.

С уважением. "ИНФОРКОМ"