«ИНФОРКОМ»

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР «ZX – SPECTRUM»

Прикладная графика

Москва – 1993

«ИНФОРКОМ»

ПЕРСОНАЛЬНЫЙ КОМПЬЮТЕР

"ZX-SPECTRUM"

Прикладная графика

Москва - 1993

Цель данной книги - дать пользователю персонального компьютера ZX-Spectrum сведения об основных практических приемах использования графических возможностей компьютера в прикладных и игровых программах. Рассмотрены растровая, векторная и блочная графика. На конкретных примерах показаны приемы создания трехмерной графики.

Книга содержит большой объем конкретных примеров и практических рекомендаций и является второй частью четырехтомной серии.

"ИНФОРКОМ" приносит глубокую благодарность своему постоянному корреспонденту Алексееву А. Г. за подготовку раздела 2. 8 и Приложения 3 настоящей книги.

1. ВВЕДЕНИЕ

Уважаемый читатель! Вы держите в руках книгу, посвященную графике компьютера "ZX-Spectrum". Вероятно, Вы уже поняли, что это второй (и не последний) том нашей графической серии. Если Вам не довелось до сих пор приобрести т. 1 "Элементарная графика", это не должно Вас останавливать и не должно явиться существенным препятствием для успешной работы с "Прикладной графикой". Мы готовим наши книги как достаточно автономные издания, каждое из которых представляет самостоятельную ценность и может быть использовано для самообразования независимо от остальных.

В то же время, позвольте несколько слов уделить основным положениям, рассмотренным в предыдущем томе и основным полученным там выводам.

1.1. Три метода графического представления информации.

Если Вы взглянете на свою обширную игротеку "Синклеровских" программ, то, возможно, очень удивитесь, когда узнаете, что все многообразие графических экранов, с которым Вы можете столкнуться, укладывается в простейшую схему, включающую в себя всего лишь три метода исполнения графики:

- растровая графика;
- векторная графика;
- блочная графика.

Растровая графика характерна тем, что изображение на экране строится по точкам в соответствии с теми образами, которые заранее был созданы Вами и хранятся где-то в оперативной памяти. Эти образы часто называют шаблонами. Подготовить заранее такие образы и отгрузить их на ленту Вы можете с помощью какого-либо графического редактора. Шаблоны размером 8X8 Вы можете

подготовить и используя приведенную в Приложении 1 программу "Генератор символов" ("Charsen"). Для подготовки шаблонов размером mXn (а их часто называют "спрайтами") используют программы типа "Генератор спрайтов" (см. Приложение 2). Это же можно сделать и используя режим "Capture" широко известного графического редактора ARTSTUDIO, о чем мы расскажем в разделе 2.5.2.

С растровой графикой Вы наиболее широко сталкиваетесь в игровых программах аркадного и аркадно-адвентюрного жанра. Их легко узнать - это красочные программы типа ARMI MOVES, DIZZY, SCEPETRE OF BAGDAD. Количество экранов в этих программах, как правило, невелико - 15...30. Программисты увеличивают количество хранимых экранов за счет специальных приемов - уменьшения размеров экранного изображения до 1/3 экрана (FRANKY GOES TO HOLLYWOOD) и за счет сокращения цветовой гаммы (KNIGHT LORE, ALIEN-8). Тогда им удается увеличить количество экранов до многих десятков, а если некоторые "шаблоны" удается не хранить в памяти, а создавать программным путем непосредственно перед использованием с помощью специального "генератора", то и до нескольких сотен. Так, например, в программах очень часто генерируют всевозможные орнаментальные украшения (DUN DARACH, MARSPORT, HEAD OVER HEELS).

Векторная графика выглядит намного беднее и бледнее. Это вычисляемая графика. В памяти не хранятся готовые образы тех или иных объектов. Вместо этого там хранятся алгоритмы, с помощью которых эти объекты строятся на экране непосредственно в тот момент, когда они нужны. С такими программами Вы также должны быть знакомы. Это широко известная программа ELITE и похожие на нее STARGLIDER, ACADEMY и ин. др.

Несмотря на то, что векторная графика значительно уступает растровой по силе художественного воздействия, у нее есть огромное преимущество - экономное расходование памяти. С помощью одного и того же алгоритма, меняя входные параметры, можно получить бесконечное количество различных изображений. Исследования наших читателей показывают, например, что в программе ELITE

количество возможных галактик можно считать бесконечным, а в каждой из них по 256 звезд и у каждой Вас ожидает огромное количество неповторимых боевых ситуаций.

Векторную графику нередко используют в адвентюрных играх, ведь там необходимо значительный объем памяти уделять текстовым массивам (описанию обстановки, словарям и т.п.) и для графики остается очень мало места (THE HOBBIT, VALHALLA...). Имеется прекрасный опыт использования векторной графики в программе логико-стратегического направления - SENTINEL, где количество неповторимых задач измеряется десятками тысяч.

Мы не рассматриваем в этой книги вопросов анимации (мультипликации) - им будет посвящена следующая книга - "Динамическая графика", но надо также сказать, что динамичность изображения решается для векторной графики гораздо проще, чем для растровой. Перестроение изображения происходит более быстро и плавно. Несмотря на то, что по художественному впечатлению векторная графика уступает растровой, она позволяет использовать другие приемы психологического воздействия на пользователя - многообразие ситуаций и скорость анимации создают столь необходимый эффект присутствия и сопричастности к развивающимся на экране событиям. Высокая скорость анимации определила и еще одну широкую область использования векторной графики - для программ имитаторов (в первую очередь для авиаимитаторов и в некоторых случаях для автоимитаторов). В то же время, попытки использования векторной графики в спортивных имитаторах нельзя признать удачными.

Блочная графика выглядит наиболее простейшим решением из всех, но при творческом подходе позволяет получать очень неплохие изображения, зачастую не уступающие растровой графике. Суть ее состоит в том, что изображение на экране строится из заранее заготовленных блоков, которые по размеру обычно совпадают с размерами стандартных символов 8X8. На блочный характер графики в программах может указывать тот факт, что на экране присутствует значительное количество одинаковых элементов 8X8

или 16X16.

В блочной графике проще решаются вопросы вывода на экран. Для этого можно пользоваться стандартными процедурами ПЗУ. В памяти проще хранить большое количество изображений. Вместо растровой копии можно хранить для каждого экрана "карту" размещения блоков, которую можно выразить в виде символьной строки). Все это позволяет программисту сократить расход памяти на процедуры, обслуживающие экран, и сделать более мощный упор на логику работы программы. Художественное впечатление, получающееся при блочной графике, может быть неплохим (MANIC MINER, JET SET WILLY, BOULDER DASH) и даже весьма выразительным (DANDY, RANARAMA, AVENGER).

Как правило, блочная графика применяется для аркадных игр, не стремящихся к максимальной жизненной реальности игровых ситуаций, но она широко используется и в стратегических (OVERLORD, STALINGRAD, CONQUEST...) и в традиционных (шахматы, карты и т.п.) и в логических играх (SOCOBAN,...), т.е. в тех случаях, когда мы имеем дело с условной картой местности или с игровым полем более или менее регулярной структуры.

1.2. Три подхода к воспроизведению графики.

Мы разделили графику на три вида. Возможно, в этом есть определенный волюнтаризм, поскольку конечно же нет никаких причин, препятствующих использованию в одной программе всех трех видов одновременно, а именно так и происходит на практике. Деление это может быть и выглядит условным, ведь и в том и в другом и в третьем случае решается одна и та же задача - заполнить экранную область памяти (16384...23295) своими данными, которые воспроизведутся на экране в виде нужной Вам картинки, но вот приемы, которыми это достигается, с точки зрения техники программирования, оказываются совершенно различными - только поэтому мы рассматриваем разные виды графики отдельно.

В принципе, вся эта книга и посвящена этим приемам, но

если бы нас попросили в трех словах выразить различие в подходах к воспроизведению растровой, векторной и блочной графики, то мы это сделали бы так:

Растровая графика - копирование.
Векторная графика - рисование.
Блочная графика - печать.

Построение растрового изображения на практике сводится к КОПИРОВАНИЮ шаблона из оперативной памяти в экранную область. На БЕЙСИКе суть операций с растровой графикой выражается одной строчкой:

POKE addr_1, (PEEK addr_2) - изображение шаблона копируется из адреса addr_2 в область экрана addr_1.

На АССЕМБЛЕРе это же выглядит так:

```
LD HL, ADDR2
LD A,(HL)
LD HL, ADDR1
LD (HL),A
```

Построение векторного изображения исполняется путем изображения графических примитивов, к которым относятся: точки, линии, дуги и окружности. На БЕЙСИКе это выглядит тривиально - PLOT, DRAW, CIRCLE. На АССЕМБЛЕРе же либо пользуются соответствующими процедурами ПЗУ, либо пишут свои. Об основных приемах мы говорили в книге "Элементарная графика". Окрашивание замкнутых контуров гладким цветом или заполнение их текстурой выполняют с помощью специально подготовленных для этого процедур "заливки".

Построение блочного изображения проще всего представить, как обычную печать заранее подготовленных символов. На БЕЙСИКе это тривиально - PRINT AT. На АССЕМБЛЕРЕ используют либо процедуру RST 16, либо одну из других процедур ПЗУ, рассмотренных нами в предыдущей книге, либо пишут свою процедуру, опирающуюся на RST 16.

1.3. Три основные проблемы.

С точки зрения программирования в машинном коде при работе с графикой есть три основные проблемы, которые приходится решать, независимо от того, с каким методом воспроизведения графики Вы имеете дело. Они связаны с установлением соответствия между координатами изображения на экране и адресом в экранной области памяти. Грубо говоря, как найти адрес в экранной области памяти для заданной экранной координаты?

Почему же мы говорим о трех проблемах, а не об одной? Дело в том, что на "Спектруме" и графика низкого разрешения и графика высокого разрешения прекрасно уживаются на одном экране, поэтому эту задачу приходится решать трижды:

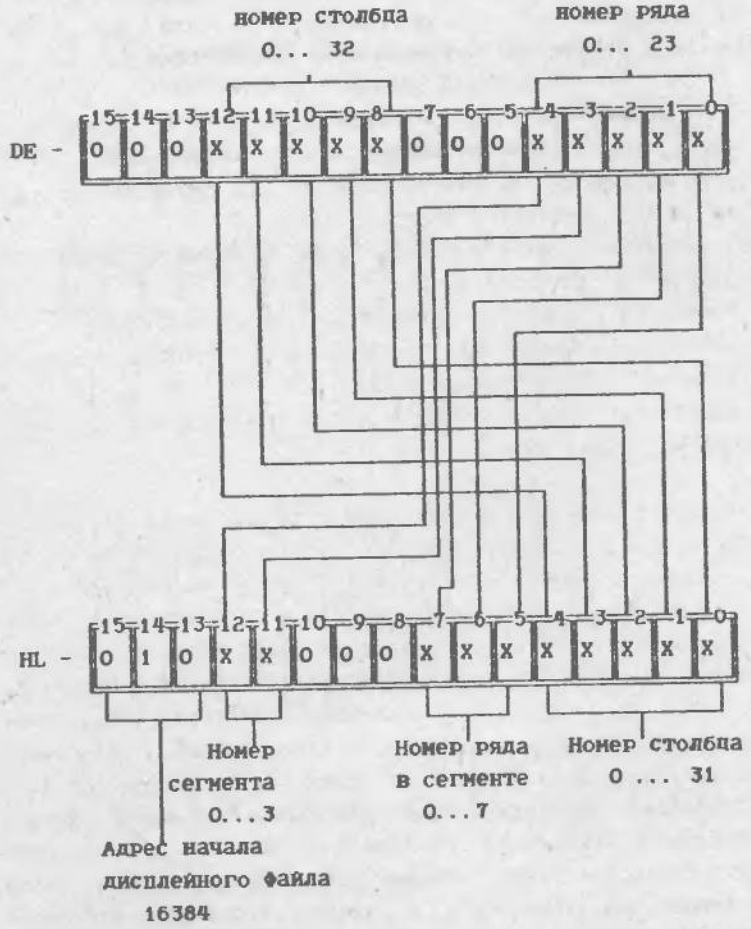
- определение адреса в памяти, если экранные координаты заданы в знакоместах;
- определение адреса в файле атрибутов по заданным координатам (координаты цветовых атрибутов могут быть заданы только в знакоместах).
- определение адреса в памяти, если экранные координаты заданы в пикселах;

Подробному разбору соответствующих приемов и алгоритмов мы посвятили первый том нашей серии, но кратко будет нелишним их повторить для тех, кто его не читал или читал, но немного подзабыл. Давайте будем условно считать, что экранные координаты заданы в регистровой паре DE, а искомый адрес будем формировать в регистровой паре HL. Причем в регистре D задаем координату X, а в регистре E - координату Y. Исключением является случай, когда координата задана в пикселах. В этом случае байт памяти указывает не на точку экрана, а на линию в знакоместе. Для того, чтобы указать на точку, надо еще определить номер бита в байте. Для этого привлекаем регистр B, в котором и определяем номер бита в искомом байте, который должен быть включен, чтобы точка появилась на экране точно в заданной координате. Использование для этой цели именно регистра B имеет тот смысл, что его сразу же можно использовать в качестве счетчика для организации с помощью команды DJNZ отсчета бита, подлежащего включе-

НИЮ.

A: Расчет адреса в дисплейном файле
(координаты заданы в знакоместах).

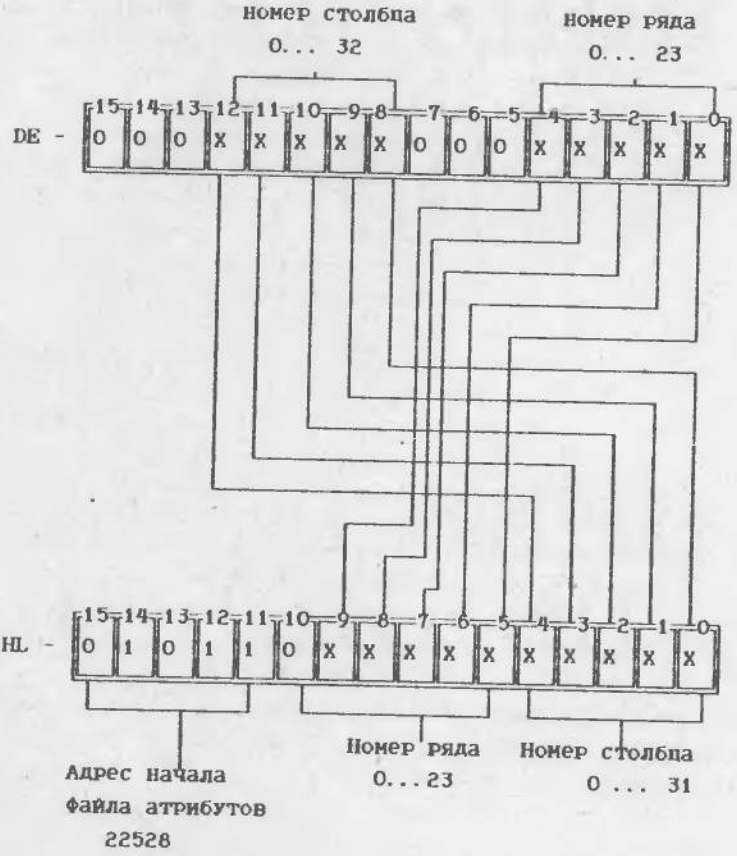
Структурная схема:



Алгоритм GET_ADR_S

```
LD A,E      LD A,E      RRA         LD L,A
AND 18H     AND 07      RRA
OR 40H      OR A        RRA
LD H,A      RRA         ADD A,D
```

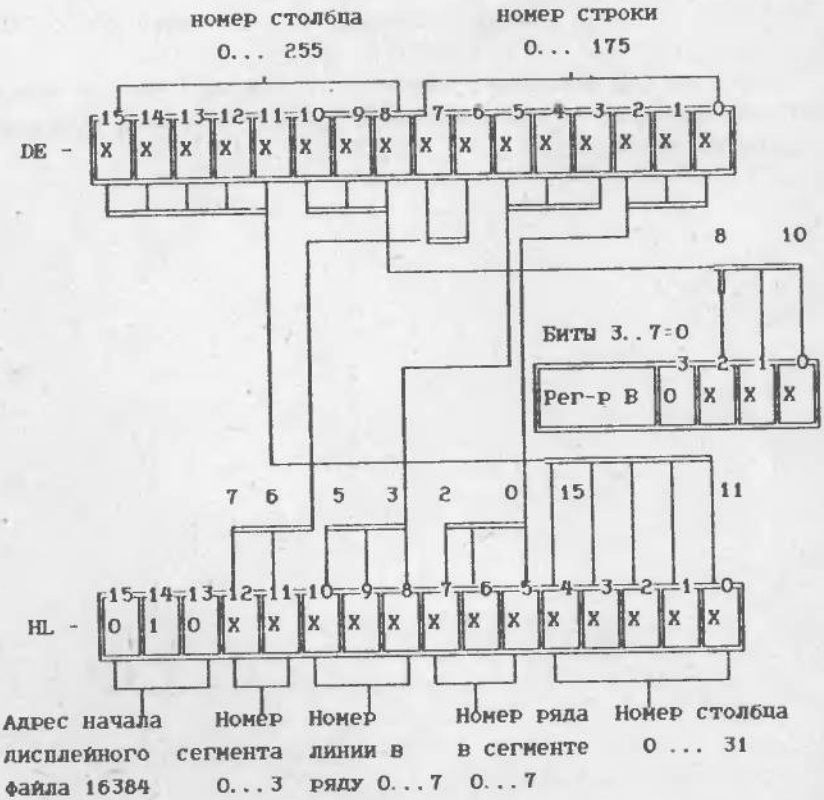
Б: Расчет адреса в файле атрибутов
(координаты заданы в знакоместах).



Алгоритм GET_ADR_A

```
LD A,E      SRL A       AND 07      RRA
AND 18H     OR 58H      OR A        RRA
SRL A       LD H,A      RRA         ADD A,D
SRL A       LD A,E      RRA         LD L,A
```

B: Расчет адреса в дисплейном файле
(координаты заданы в пикселах).



Алгоритм GET_ADR_P

```
LD A,AFH    RRA         RLCA        AND 07
SUB E       XOR E       XOR E       LD B,A
LD E,A      AND 0F8H    AND C7H     LD A,08
AND A       XOR E       XOR E       SUB B
RRA         LD H,A      RLCA        LD B,A
SCF         LD A,D      RLCA
RRA         RLCA        LD L,A
AND A       RLCA        LD A,D
```

На этом мы закончим повторение материала первого тома и перейдем непосредственно к методике работы с графикой в пользовательских программах.

2. РАСТРОВАЯ ГРАФИКА

При работе с растровой графикой программисту приходится решать следующие задачи:

1. Выбрать формат, в котором шаблоны будут храниться в Вашей программе. Может быть, Вы примете решение о том, что все шаблоны будут у Вас иметь одинаковый размер. Если же Вы решите, что различные шаблоны могут иметь различные размеры, то очевидно, что вместе с картой битов шаблона Вам надо будет сохранять и данные о его размерах по горизонтали и вертикали. Иногда оказывается важным вместе с изображением сохранять и координаты левого верхнего угла окна экрана, из которого оно было взято. Тогда его можно будет при восстановлении на экране поместить точно на свое место.

Здесь же надо решить для себя, как поступить с цветовыми атрибутами. То ли Вы будете хранить в памяти только битовую карту своего изображения, а назначение цвета выполнять при выдаче его на экран, то ли Вы будете хранить данные по атрибутам вместе со своими шаблонами. При этом следует помнить, что раз минимальный размер квадрата атрибутов на "Спектруме" равен 8X8, то данные по атрибутам целесообразно хранить только для таких шаблонов, вертикальный и горизонтальный размер которых кратен 8, например 16X24, 32X32 и т.п. Приняв решение о совместном хранении атрибутов, Вы также ограничите себя в способах выдачи Ваших шаблонов на экран. Перед тем, как изобразить шаблон на экране, Вам надо будет четко рассчитать координату положения его левого верхнего угла с тем, чтобы Ваше изображение точно вписалось в квадраты сетки низкого разрешения экрана (32X24), иначе если хоть какая-то часть изображения выйдет за пределы своих знакомест, произойдет переключение цветовых атрибутов в соседних знакоместах и могут нарушиться цвета картинки, ранее имевшейся на экране. Это явление называется "клэшингом" атрибутов. Всякий, кто работал с цветом в графическом редакторе "ARTSTUDIO", конечно же сталкивался с этим явлением и знает, насколько оно неприятно.

Радикально с клэшингом ничего поделать нельзя, ведь это системная особенность "Спектрума", связанная с тем, что только монохромная информация может воспроизводиться на экране в высоком разрешении, а все атрибуты воспроизводятся с низким разрешением. Но некоторые рекомендации по уменьшению вредного воздействия клэшинга все же могут быть даны и мы это сделаем.

2. Подготовить шаблон (шаблоны) изображения с помощью графического редактора или какого-либо другого программного средства.

3. Отгрузить шаблон (шаблоны) на внешний носитель.

4. Организовать свои шаблоны в библиотеку (таблицу) в соответствии с избранным форматом. Это нужно для того, чтобы впоследствии Ваша программа могла быстро находить в памяти данные для шаблона по его номеру. Если все шаблоны в Вашей программе имеют одинаковые размеры, то проблем с выбором N-ного шаблона из таблицы не будет, но если у Вас шаблоны произвольные, да к тому же их битовая карта хранится вместе с атрибутами, то придется решить вопрос о том, как отделять шаблоны друг от друга.

Вряд ли целесообразно применять какие-либо маркеры, отделяющие один шаблон от другого, как это делается в текстовых таблицах, где инвертируют последний байт в каждом отдельном сообщении. Там этот байт легко распознается, поскольку его значение после инверсии получается больше 128, а никакой стандартный символ иметь код больше 128 не может. Здесь никакой маркер распознать не удастся, т. к. любой байт может оказаться не маркером, а графическими данными. Скорее всего, Вы примете решение о том, что перед каждым шаблоном надо поставить один-два префиксных байта, содержащих длину шаблона вместе с атрибутами и, кстати, вместе с этими двумя байтами.

Возможно, Вам придется написать свою служебную программу "Библиотекарь", которая соберет с ленты отгруженные Вами картинки, измерит их длину, предложит Вам ввести атрибуты для каждого знакоместа и расположит картинки в памяти одну за другой

вместе с префиксными байтами в виде единого цельного файла.

5. Может быть, Вы предполагаете использовать в программе очень много графики. Тогда рано или поздно перед Вами встанет вопрос о необходимости компрессии (архивации) Ваших изображений и чем более крупные шаблоны Вы применяете, тем скорее наступит этот момент. Придется написать программу "Архиватор", которая сожмет размеры файла шаблонов. Поскольку никакой скорости от нее не требуется, она вполне может быть написана и на БЕЙСИКе.

6. Подготовить процедуру для сканирования по файлу шаблонов, которая по номеру нужного Вам шаблона найдет его начало.

7. Подготовить процедуру для дезархивации шаблона и печати его на экране. Здесь опять же нужно определиться, как Вам лучше дезархивировать картинку - сразу на экран или сначала в отведенный для этой цели буфер, а потом оттуда перебрасывать ее на экран. В чем здесь разница, Вы наверное догадались и сами. Если сделать буфер, то увеличится размер использованной оперативной памяти, но зато можно очень быстро производить освежение экрана. Если же дезархивацию выполнять прямо на экран, то экономится память, но процесс печати естественно замедлится. Так что Вы сами должны решить, что для Вашей программы важнее - быстродействие или объем. Во многих случаях имеет смысл в качестве буфера иметь где-то в оперативной памяти полную копию экрана. Это тем более целесообразно, что процедуры для печати на экран и в буфер фактически одни и те же, различия наиминимальнейшие.

Такова последовательность действий в самом общем случае. На практике же каждый этап требует глубокого осмысления и многочисленных экспериментов. Ставя один опыт за другим, оттачивая наилучшие решения, постепенно, программисты и приходят к тем замечательным программам, с которыми конечно же знаком каждый "синклерист".

2.1. Печать шаблона 8X8.

Это, конечно, наипростейшая задача. Предположим, что нам нужно поместить шаблон с порядковым номером MM на экран в позицию с координатами X и Y. Пусть таблица шаблонов расположена в оперативной памяти, начиная с адреса ADDR_T. Здесь может быть разница в том, как заданы координаты X и Y. Если они заданы в знакоместах, то напечатать шаблон на экране надо с привязкой к знакоместу и тогда печать производится КОПИРОВАНИЕМ БАЙТОВ шаблона в область экрана. То есть шаблон копируется по восьмипиксельным линиям, каждую из которых представляет один байт. Если же координаты заданы в пикселах, то тогда печать производится КОПИРОВАНИЕМ БИТОВ шаблона в область экрана.

Операция выполняется в три приема.

На первом этапе номер (индекс) нужного нам шаблона устанавливается в регистре L. Если у нас больше 256-ти шаблонов, то в регистровой паре HL. В регистровой паре DE устанавливаем базовый адрес таблицы и ищем искомый результат в регистровой паре HL.

```
LD DE,ADDR_T  ; Базовый адрес таблицы шаблонов.
LD HL,MM      ; Номер нашего шаблона.
ADD HL,HL     ; Умножили его на 2.
ADD HL,HL     ; Умножили его на 4.
ADD HL,HL     ; Умножили его на 8.
ADD HL,DE     ; Нашли адрес начала нашего шаблона в HL.
PUSH HL       ; Запомнили его на стке до востребования.
```

На втором этапе по координатам X и Y, установленным в регистровой паре DE находим соответствующий им адрес в дисплейном файле. При этом возможны два варианта. В первом случае координаты заданы в знакоместах, а во втором случае - в пикселах. И в том и в другом случае можно воспользоваться процедурами, которые были нами подробно рассмотрены в книге "Элементарная графика" и кратко повторены в начале этой книги (раздел 1.3).

```
LD D,X
LD E,Y
CALL GET_ADR_S  ; См. 1.3
     или
CALL GET_ADR_P  ; См. 1.3
```

На третьем (заключительном) этапе выполняется операция копирования данных из одной области памяти в другую.

A: Координаты заданы в знакоместах.

Если координаты заданы в знакоместах, то задача сводится к копированию 8-ми байтов и нам достаточно одного цикла на 8 шагов.

```
        POP DE      ; Взяли со стека адрес нашего шаблона.
        LD B,8      ; Организация цикла в 8 шагов.
LOOP    LD A,(DE)   ; Взяли байт из шаблона.
        LD (HL),A   ; Поместили его на экран.
        INC DE      ; Переход к очередному байту шаблона.
        INC H       ; Переход к очередной линии
                    ; знакоместа экрана.
        DJNZ,LOOP   ; Если не все 8 байтов скопированы,
                    ; то возврат на LOOP.
```

B: Координаты заданы в пикселах.

Во втором случае, когда координаты заданы в пикселах, приходится решать вопрос о побитном копировании каждого байта и в этом случае нам уже необходимы два цикла - внутренний и внешний, каждый на восемь шагов. Благодаря тому, что координаты заданы в пикселах, мы можем использовать для печати точек на экране встроенную в ПЗУ процедуру PLOT (22E5H = 8933). Надо только обеспечить наличие координаты X в регистре B, а координаты Y - в регистре C и не забыть, что при своей работе эта процедура "портит" регистровую пару HL. PLOT относится к быстроработающим процедурам ПЗУ и ее применение в Ваших программах почти всегда можно считать целесообразным. Тогда алгоритм сводится к тому,

что наш шаблон сканируется по восьми рядам и в тех местах, где бит включен, вызывается процедура PLOT для печати точки на экране, а там, где бит выключен, этот вызов не происходит. Сканирование производится вращением байта шаблона через флаг переноса. При этом если бит включен, то включается флаг переноса, что и служит индикатором необходимости вызова PLOT.

```
        POP HL          ;Взяли со стека адрес шаблона.
        LD E,Y          ;Координата Y левого верхнего угла.
        LD C,8          ;Счетчик линий шаблона.
LOOP_2  LD D,X          ;Координата X левого верхнего угла.
        LD A,(HL)       ;Очередной копируемый байт.
        OR A            ;Принудительный сброс флага переноса.
        LD B,8          ;Счетчик пикселов в линии.
LOOP_1  PUSH BC         ;Освободили BC для работы с процедурой
                        ;PLOT.
        LD B,D          ;Координаты левого
        LD C,E          ;верхнего угла.
        RLA             ;Вращение влево через флаг переноса.
        JR NC,PASS      ;Если бит погашен, то вызывать PLOT
                        ;не надо.
        PUSH HL         ;Перед вызовом PLOT надо сохранить HL.
        CALL 22E5H      ;Вызов процедуры ПЗУ PLOT.
        POP HL          ;Восстановление HL
PASS    INC D           ;Переход к новой координате X.
        POP BC          ;Восстановление счетчиков.
        DJNZ LOOP_1     ;Если не все восемь битов в байте
                        ;проверены, возврат на LOOP_1.
        DEC C           ;Переход к новой координате Y.
        INC HL          ;Переход к очередному байту шаблона.
        LD A,B          ;
        OR C            ;Проверка регистра C на ноль.
        JR NZ LOOP_2    ;Если не все восемь байтов просмотре-
                        ;ны, следует возврат на LOOP_2 для
                        ;повтора.
        RET             ;Выход из процедуры.
```

В заключение описания этого алгоритма мы должны высказать одно замечание. Дело в том, что здесь ничего не сделано для проверки координат позиции печати ни по X, ни по Y, а вообще-то надо проверять, а ложатся ли они в пределы экрана? Здесь этот момент опущен для того, чтобы освободить алгоритм от излишних подробностей, не имеющих отношения к обсуждаемому вопросу.

2.2. Печать шаблона произвольного формата без атрибутов.

2.2.1. Координаты заданы в знакоместах.

Для печати произвольного шаблона размером (m X n) знакомест мы можем воспользоваться тем же алгоритмом, которым печатали шаблон 8 X 8 пикселов, повторив его m раз по вертикали и n раз по горизонтали. Таким образом, оказывается, что там, где для печати шаблона 8 X 8 достаточно было одного цикла в 8 шагов, теперь необходимы три вложенных цикла:

```
LOOP_3   - n шагов
LOOP_2   - m шагов
LOOP_1   - 8 шагов
.....
.....
.....
```

Порядок построения шаблона оказывается таким:

1	.	.	.
2	.	.	.
3	.	.	n-2
.	.	.	n-1
.	.	.	n

Рис. 1

Для работы процедуры нам понадобятся несколько программных переменных.

```
ADDR_S    DEFW 0000        ;Адрес в дисплейном файле.
COORDS    DEFW 0000        ;Координаты X и Y.
NUMBER    DEFB 00          ;Номер шаблона.
BITMAP    DEFW 0000        ;Адрес начала файла шаблонов.
LENGTH    DEFB 00          ;Длина стандартного шаблона.
SIZE_X    DEFB 00          ;Размер шаблона по горизонтали
                           ;(в знакоместах).
SIZE_Y    DEFB 00          ;Размер шаблона в знакоместах
                           ;по вертикали.

          LD DE,(COORDS)   ;Ввод координат левого верхнего
                           ;угла;
          LD A,E
          AND 18H
          OR 40                Расчет адреса в дисплейном
          LD H,A               файле по координате, задан-
          LD A,E               ной в знакоместах.
          AND 07
          OR A
          RRA
          RRA
          RRA
          RRA
          ADD A,D
          LD L,A
          LD (ADDR_S),HL   ;Запомнили адрес в дисплейном
                           ;файле.
          LD A,(NUMBER)    ;Номер шаблона.
          LD B,A           ;Организовали счетчик шаблонов.
          LD DE, LENGTH    ;Длина шаблона
          LD HL, BITMAP    ;Начало файла шаблонов.
LOOP      ADD HL,DE        ;Смещение на длину шаблона.
          DJNZ LOOP        ;Повторяем столько раз, каков
                           ;порядковый номер шаблона.
```

```
        LD DE,(ADDR_S)    ;Адрес в дисплейном файле.
        LD B,SIZE_Y       ;Счетчик знакомест по вертикали.
        LD C,SIZE_X       ;Счетчик знакомест по горизонтали.
LOOP_3  PUSH DE           ;Сохранили адрес экрана.
LOOP_2  PUSH BC           ;Сохранили счетчики на стеке.
        LD C,8            ;Организовали счетчик линий в
                          ;знакоместе.
        LD B,8            ;Организовали счетчик пикселов в
                          ;текущей линии знакоместа.
        PUSH DE           ;Сохранили экранный адрес
                          ;еще раз.
LOOP_1  LD A,(HL)         ;Загрузили в аккумулятор
                          ;очередной байт шаблона.
        LD (DE),A         ;Перенос текущего байта из
                          ;оперативной памяти в экранную.
        INC D             ;Переход к следующей линии
                          ;знакоместа на экране снизу.
        INC HL            ;Переход к следующему байту
                          ;шаблона.
        DJNZ LOOP_1       ;Конец цикла копирования шаблона
                          ;8 X 8.
        POP DE            ;Восстановили экранный адрес.
        LD A,20H
        ADD A,E           ;Переход к следующему экранному
        LD E,A            ;ряду.
        POP BC            ;Восстановили счетчики знакомест.
        DJNZ LOOP_2       ;Если не все знакоместа по
                          ;вертикали скопировали, возврат
                          ;назад.
        POP DE            ;
        INC E             ;Переход к следующему знакоместу
                          ;справа.
        DEC C             ;Уменьшили счетчик столбцов.
        LD A,C            ;Проверка счетчика
        OR B              ;на ноль.
        JR NZ,LOOP_3      ;Если не весь шаблон скопирован,
                          ;возврат на LOOP_3.
        RET
```

2.2.2. Координаты заданы в пикселах.

Как и в случае с копированием шаблона 8X8, здесь встает задача побитного копирования и наилучшим способом, как мы уже говорили, является использование встроенной в ПЗУ процедуры PLOT. С точки зрения удобства программирования, есть принципиальное отличие от случая, когда координаты заданы в знакоместах. Оно состоит в порядке копирования байтов шаблона (сравните рис. 1 и рис. 2):

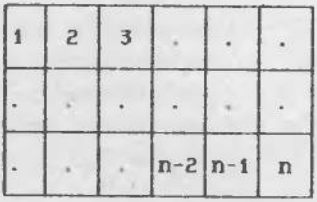


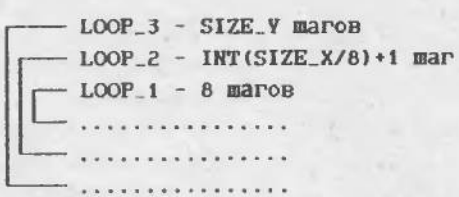
Рис. 2

Есть и еще одно отличие от случая, когда координаты заданы в знакоместах. Здесь нам не надо определять адрес в дисплейном файле, соответствующий координатам левого верхнего угла, поскольку в стандартной процедуре PLOT уже все для этого есть, если мы предварительно задали координаты в регистровой паре BC.

Программные переменные.

```
COORDS    DEFW 0000      ;Координаты X и Y в пикселах.
NUMBER    DEFB 00        ;Номер шаблона.
BITMAP    DEFW 0000      ;Адрес начала файла шаблонов.
LENGTH    DEFB 00        ;Длина стандартного шаблона.
SIZE_X    DEFB 00        ;Размер шаблона по горизонтали
                         ;(в пикселах).
SIZE_Y    DEFB 00        ;Размер шаблона в пикселах
                         ;по вертикали.
```

Здесь нам также достаточно трех вложенных циклов:



Внешний цикл LOOP_3 выполняет копирование строк шаблона. Длина цикла равна высоте шаблона по вертикали. Промежуточный цикл LOOP_2 копирует отдельную строку шаблона по восьмипиксельным линиям. Количество этих линий равно числу полных и неполных знакомест. которые занимает шаблон по горизонтали (оно вычисляется). Внутренний цикл копирования одной восьмипикселной линии состоит из восьми шагов вращения байта шаблона и печати с помощью процедуры PLOT точки на экране в тех случаях. когда соответствующий пиксел в шаблоне включен.

```
        LD A,(NUMBER) ;Номер шаблона.
        LD B,A        ;Организовали счетчик шаблонов.
        LD DE, LENGTH ;Длина шаблона
        LD HL, BITMAP ;Начало файла шаблонов.
LOOP    ADD HL,DE     ;Смещение на длину шаблона.
        DJNZ LOOP     ;Повторяем столько раз, каков
                      ;порядковый номер шаблона.
        LD DE,(COORDS);Координаты левого верхнего угла:
                      ;E - координата Y;
                      ;D - координата X.
        LD C,SIZE_Y   ;Счетчик линий по вертикали.
        LD B,SIZE_X   ;Размер по горизонтали в пикселах.
        LD A,B
        AND 0F8H      ;Гашение трех младших битов.
        CP B          ;Проверяем не делится ли SIZE_X
                      ;на 8 нацело.
        RRCA          ;Три раза вращение вправо
        RRCA          ;эквивалентно делению на восемь.
        RRCA          ;То же самое можно сделать и
                      ;командой сдвига SRL, но она может
```

```
                        ;влиять на флаг Z, а у нас он хранит
                        ;результат операции CP B и портить
                        ;его не надо.
          JR Z, PASS    ;Если SIZE_X делится на 8 нацело,
                        ;то прибавлять единицу не нужно.
          INC A         ;INT (X_SIZE/8) + 1
PASS      LD B,A        ;Счетчик цикла по горизонтали, т.е.
                        ;размер по горизонтали в знакоместах
                        ;(как в полных, так и не в полных).
LOOP_3    PUSH BC       ;Сохранили счетчики на стеке.
  LOOP_2  PUSH BC       ;Сохранили счетчики на стеке.
          LD A,(HL)     ;Очередной копируемый байт.
          OR A          ;Принудительный сброс флага переноса.
          LD B,8        ;Счетчик пикселов в линии.
    LOOP_1 PUSH BC      ;Освободили BC для работы с
                        ;процедурой PLOT.
          LD B,D        ;Координаты левого
          LD C,E        ;верхнего угла.
          RLA           ;Вращение влево через флаг переноса.
          JR NC,PASS_1  ;Если бит погашен, то вызывать PLOT
                        ;не надо.
          PUSH HL       ;Перед вызовом PLOT сохраним HL.
          CALL 22E5H    ;Вызов процедуры ПЗУ PLOT.
          POP HL        ;Восстановление HL
  PASS_1  INC D         ;Переход к новой координате X.
          POP BC        ;Восстановление счетчика пикселов.
          DJNZ LOOP_1   ;Если не все восемь битов в байте
                        ;проверены, возврат на LOOP_1.
          POP BC        ;Восстановление счетчика линий.
          DJNZ LOOP_2   ;Если не все линии в ряду
                        ;скопированы, возврат на LOOP_2.
          POP BC        ;Восстановление счетчика линий.
          DEC C         ;Если не все линии по вертикали
          JR NZ,LOOP_3  ;скопированы, то возврат на LOOP_3.
          RET           ;Возврат.
```

2. 3. Печать произвольного шаблона с атрибутами.

Само собой разумеется, что когда речь идет о печати шаблона вместе с его цветовыми атрибутами, речь должна идти о печати шаблонов, размеры которых заданы в знакоместах, то есть кратны 8, т. к. атрибуты задаются только с привязкой к знакоместам. Итак, предположим, что нам надо напечатать на экране заранее приготовленные шаблоны, имеющие размеры m знакомест по вертикали и n знакомест по горизонтали. В самом общем случае размеры m и n у различных шаблонов могут быть и различными. Таким образом, встает вопрос о том, как хранить информацию о размере K-го шаблона и о его цветовых атрибутах, т. е. встает вопрос о выборе формата шаблона.

2. 3. 1. Выбор формата шаблона.

Выбор формата, в котором Вы храните данные, необходимые для успешной работы Вашей программы - это очень важный творческий процесс. Нерационально выбранный формат может очень сильно замедлить скорость работы программы и привести к излишнему расходу памяти. Более того, как показывает практика программирования, в большинстве случаев структура грамотно составленной программы (рабочих процедур) оказывается очень сильно связанной со структурой данных, которые этими процедурами обрабатываются. Так что непродуманный выбор формата данных может привести и к еще более неприятному последствию, когда программист путается в своих процедурах, теряет логику работы программы и, запутавшись, бросает едва начатое дело.

Существует много различных типов структур данных, которые могут применяться в практической работе. Вам, конечно известна такая структура, как "стек", когда данные закладываются сверху и сверху же и извлекаются по принципу "последним пришел, первым уйдешь". В книге "Элементарная графика" в разделе 3. 11 мы рассиатривали такую структуру, как "конвейер", существует еще стркутура, называемая "очередь", отличающаяся от "стека" тем, что данные закладываются сверху, а извлекаются снизу по прин-

типу "последним пришел, последним уйдешь". Но все это структуры для работы с динамически меняющимися данными, а графические шаблоны, размещаемые в памяти заранее, очевидно к таким не относятся. Для них очень хорошо подходит организация в таблицы и векторы.

Вариант 1.

Что такое таблицы данных очевидно знают все. Это массивы, которые бывают двумерными, трехмерными и т.д. Многих начинающих программистов пугает неуклюжее слово "вектор", которое нередко встречается в текстах пояснений к программам. Со школы все помнят, что "вектор" - это отрезок прямой, для которого кроме длины зачем-то важно знать еще координаты начала и конца и направление. К нашим векторам данных это школьное понятие хоть и имеет косвенное отношение, но такое далекое, что лучше о нем и не вспоминать. Дело пойдет гораздо проще, если сказать, что вектор - это простой и всем понятный одномерный массив данных. В этом смысле и символьная строка - тоже как бы вектор. Если таблица - это набор строк и столбцов, то любая строка в ней - вектор и любой столбец - тоже вектор.

Таким образом, вектор данных - это просто одномерная упорядоченная структура. Иногда в программах употребляют векторы-указатели. Это тоже одномерные массивы, каждый элемент которых на что-то указывает. Например, Вы храните в программе 500 различных шаблонов, каждый из которых имеет разную длину. Первый шаблон начинается с адреса 40000, второй - с 40025-го, третий - с 40157-го и т.п., тогда массив из 500 чисел:

40000, 40025, 40157 ...

и будет вектором-указателем на Вашу таблицу шаблонов, а разместить его Вы можете где угодно, например начиная с базового адреса 39000, оставив 1000 байтов на 500 чисел (по два байта на каждое).

Итак, вот Вам первый способ хранения шаблонов в памяти компьютера. У него есть очевидные достоинства и недостатки. К достоинствам можно отнести то, что это, по-видимому, самый

быстрый способ отыскания адреса, с которого начинается Ваш шаблон.

```
        LD DE,ADDR_1 ;Базовый адрес вектора указателя.
        LD HL,NUMBER ;Номер нужного Вам шаблона.
        ADD HL,HL    ;Умножили его на 2 (по 2 байта на
                     ;каждый адрес).
        ADD HL,DE    ;Установили указатель на адрес
                     ;нужного Вам шаблона.
        LD A,(HL)    ;  Стандартный прием исполнения
        INC HL       ;  команды LD HL,(HL),
        LD H,(HL)    ;  которой к сожалению нет
        LD L,A       ;  в процессоре Z-80.
```

Если Вы не собираетесь хранить в программе графические образы по размеру большие, чем 5X5 (6X4, 3X9) знакомест, то расход памяти на вектор-указатель можно сократить в два раза, отводя не по два байта на хранение каждого адреса начала шаблона, а только по одному байту на хранение его длины (при малых размерах шаблона она уложится в один байт). Тогда способ отыскания адреса начала шаблона будет таким:

```
        LD DE,ADDR_1 ;Базовый адрес вектора указателя.
        LD HL,ADDR_2 ;Базовый адрес таблицы шаблонов.
        LD BC,NUMBER ;Номер нужного Вам шаблона.
        LD A,C       ;  Поменяли местами регистры
        LD C,B       ;  C и B, поскольку счетчик
        LD B,A       ;  младшего разряда удобнее
                     ;  делать в B.
LOOP    LD A,(DE)    ;Взяли длину шаблона в аккумулятор.
        INC DE       ;Переход к следующей позиции вектора-
                     ;указателя.
        OR A         ;Сбросили флаг переноса.
        ADD A,L      ;Прибавили к HL
        LD L,A       ;длину шаблона.
        JR NC,PASS   ;Если аккумулятор переполнился,
        INC H        ;то увеличиваем старший регистр HL.
PASS    DJNZ LOOP    ;Если младший разряд счетчика не
```

```
                ;исчерпан, возврат на LOOP.
LD A,C          ;  Стандартный прием проверки
OR B            ;  старшего разряда счетчика
RET Z           ;  на ноль.
DEC C           ;Уменьшили старший разряд.
JR LOOP         ;Повтор вычислений.
```

К недостаткам на первый взгляд хочется отнести дополнительный расход памяти для хранения вектора-указателя, но это не совсем верно. Дело в том, что если не строить вектор-указатель, то пришлось бы вместе с каждым шаблоном хранить данные по его длине и это заняло бы ничуть не меньший объем памяти.

Более очевидный недостаток - сложность в подготовке программы к работе и в ее отладке. Незначительные изменения в шаблонах будут приводить к тому, что "поплывут" все адреса и вектор-указатель надо будет переделывать заново.

Конечно, самой радикальной рекомендацией было бы использовать все шаблоны в программе только одного размера, тогда многое бы упростилось, уменьшились бы размеры расходуемой памяти и возросла бы скорость работы программы, а именно к этому Вы и должны стремиться.

Тогда вроде бы и нет смысла употреблять такую структуру и мы не стали бы о ней и говорить, но есть один случай, когда она может оказаться весьма и весьма полезной. Дело в том, что с точки зрения расхода памяти хранить в программе все графические образы - дело очень расточительное. Рано или поздно программист приходит к тому, что их надо архивировать путем компрессии (как это делают, мы поговорим в свое время). И в этом случае даже если у Вас первоначально все шаблоны имели равную длину, то после компрессии их размеры могут оказаться какими угодно. Вместо упорядоченной таблицы шаблонов Вы получите беспорядочный набор данных и там вектор-указатель может Вам очень пригодиться. При этом сложность в подготовке вектора-указателя и в отладке программы парируют тем, что рутинную работу по исполнению вектора поручают самой компрессирующей программе (а это легко

сделать, поскольку она-то имеет дело с шаблонами одинаковой длины).

Вариант 2.

Во втором варианте мы обойдемся без вектора-указателя и рассмотрим такой формат шаблонов, при котором все необходимые данные хранятся вместе с самим шаблоном. Опять же будем предполагать, что речь идет о хранении в памяти произвольных шаблонов различной длины. Решим для себя вопрос о том, в каком порядке будут храниться байты, представляющие из себя раскладку шаблона. Например так, как показано на рис. 3.

1	m+1	.	.
2	m+2	.	.
3	m+3	.	m*n-2
.	.	.	m*n-1
m	m*2	.	m*n

Рис. 3

Затем разработаем для себя формат шаблона, например такой:

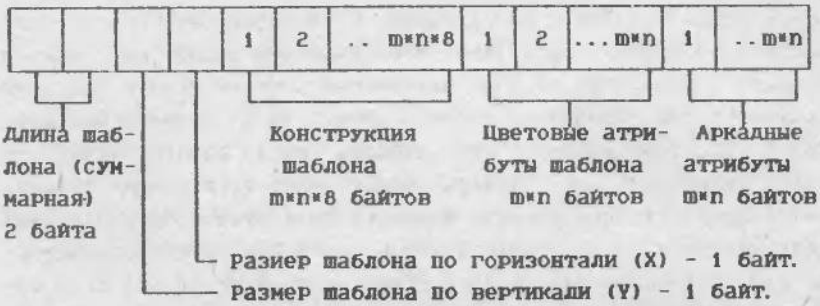


Рис. 4

В первых двух байтах (а может быть Вам достаточно и одного байта) хранится суммарная длина шаблона, включая сюда все атрибуты, а также и сами эти два контрольных байта. За ними идут два байта, указывающие на размер шаблона по горизонтали и по вертикали. Затем m*n*8 байтов занимает черно-белая конструкция шаблона. Ясно, что на каждое знакоместо нам надо отвести по 8 байтов. На цветовые атрибуты отводим столько байтов, сколько знакомест занимает Ваш шаблон - m*n байтов. Если Вы создете игровую программу, то Ваши шаблоны могут иметь и дополнительные аркадные атрибуты. О том, что это такое, мы расскажем чуть позже, а пока скажем, что и под них нужно предусмотреть место. Размер этого места целиком зависит от Вашей фантазии, но для примера предположим, что тоже по одному байту на каждое знакоместо.

Итак, один шаблон может иметь размер m*n*10+4 байта.

Сканирование по файлу шаблонов организовать несложно:

```
        LD HL,ADDR_1 ;Базовый адрес таблицы шаблонов.
        LD BC,NUMBER ;Номер нужного Вам шаблона.
        LD A,C       ;┐ Поменяли местами регистры
        LD C,B       ;│ C и B, поскольку счетчик
        LD B,A       ;│ младшего разряда удобнее
                     ;┘ делать в B.
LOOP    LD E,(HL)    ;Загрузили в E младший байт длины.
        INC HL       ;Перешли к старшему байту.
        LD D,(HL)    ;Загрузили в D старший байт.
        DEC HL       ;Скорректировали нежелательное
                     ;последствие команды INC HL.
        ADD HL,DE    ;Нашли начало очередного шаблона.
        DJNZ LOOP    ;Проверили младший разряд счетчика
                     ;шаблонов на ноль.
        LD A,C       ;┐ Стандартный прием проверки
        OR B         ;│ старшего разряда счетчика
        RET Z        ;┘ (регистр C) на ноль.
        DEC C        ;Уменьшили старший разряд.
        JR LOOP      ;Повтор вычислений.
```

В результате работы этой процедуры в регистровой паре HL будет установлен адрес, с которого начинается нужный нам шаблон, хотя надо помнить, что это еще не сам шаблон, а первые четыре служебных байта (два байта на длину и два байта на его размер).

2.3.2. Печать шаблона с цветовыми атрибутами.

Итак, мы определились с форматом, в котором в оперативной памяти компьютера хранится файл шаблонов и написали процедуру, которая по заданному номеру шаблона найдет его начало в этом файле. Следующая задача - напечатать этот шаблон вместе с атрибутами в заданных координатах левого верхнего угла (координаты заданы в знакоместах).

Исходное состояние: адрес начала шаблона в регистровой паре HL:

Процедура печати шаблона с атрибутами состоит как бы из двух частей. На первом этапе печатается черно-белая конструкция шаблона, а на втором этапе на нее накладываются цветовые атрибуты.

```
ADDR_S    DEFW 0000        ;Адрес в дисплейном файле.
ADDR_A    DEFW 0000        ;Адрес в файле атрибутов.
COORDS    DEFW 0000        ;Координаты X и Y.
NUMBER    DEFB 00          ;Номер шаблона.
BITMAP    DEFW 0000        ;Адрес начала файла шаблонов.
ATRMAP    DEFW 0000        ;Адрес начала атрибутов.
SIZE_X    DEFB 00          :Размер шаблона по горизонтали
                           ;(в знакоместах).
SIZE_Y    DEFB 00          ;Размер шаблона в знакоместах
                           ;по вертикали.

          INC HL           ;Пропускаем два байта, хранящие
          INC HL           ;длину шаблона.
          LD A,(HL)        ;Высота шаблона (Y).
          LD SIZE_Y,A      ;
```

```
        INC HL              ;
        LD A,(HL)           ;Ширина шаблона (X).
        LD SIZE_X,A
        INC HL              ;Теперь HL указывает на начало
        LD BITMAP,HL        ;графической информации шаблона.
        LD DE,(COORDS)      ;Координаты X и Y.
        LD A,E              |
        AND 18H             |
        OR 40               |   Расчет адреса в дисплейном
        LD H,A              |   файле по координате, задан-
        LD A,E              |   ной в знакоместах.
        AND 07              |
        OR A                |
        RRA                 |
        RRA                 |
        RRA                 |
        RRA                 |
        ADD A,D             |
        LD L,A              |

        LD (ADDR_S),HL      ;Запомнили адрес в дисплейном
                            ;файле.
        LD HL,BITMAP        ;
        LD DE,(ADDR_S)      ;Адрес в дисплейном файле.
        LD B,SIZE_Y         ;Счетчик знакомест по вертикали.
        LD C,SIZE_X         ;Счетчик знакомест по горизонтали.
LOOP_3  PUSH DE             ;Сохранили адрес экрана.
LOOP_2  PUSH BC             ;Сохранили счетчики на стеке.
        LD C,8              ;Организовали счетчик линий в
                            ;знакоместе.
        LD B,8              ;Организовали счетчик пикселов в
                            ;текущей линии знакоместа.
        PUSH DE             ;Сохранили экранный адрес
                            ;еще раз.
LOOP_1  LD A,(HL)           ;Загрузили в аккумулятор
                            ;очередной байт шаблона.
        LD (DE),A           ;Перенос текущего байта из
                            ;оперативной памяти в экранную.
```

```
        INC D           ;Переход к следующей линии
                        ;знакоместа на экране снизу.
        INC HL          ;Переход к следующему байту
                        ;шаблона.
        DJNZ LOOP_1     ;Конец цикла копирования шаблона
                        ;8 X 8.
        POP DE          ;Восстановили экранный адрес.
        LD A,20H
        ADD A,E         ;Переход к следующему экранному
        LD E,A          ;ряду.
        POP BC          ;Восстановили счетчики знакомест.
        DJNZ LOOP_2     ;Если не все знакоместа по
                        ;вертикали скопировали, возврат
                        ;назад.
        POP DE          ;
        INC E           ;Переход к следующему знакоместу
                        ;справа.
        DEC C           ;Уменьшили счетчик столбцов.
        LD A,C          ;Проверка счетчика
        OR B            ;на ноль.
        JR NZ,LOOP_3    ;Если не весь шаблон скопирован,
                        ;возврат на LOOP_3.

        LD ATRMAP,HL    ;Сейчас HL указывает в шаблоне на
                        ;начало информации по атрибутам.
        LD DE,(COORDS)  ;Координаты X и Y.
        LD A,E
        AND 18H
        SRL A
        SRL A
        SRL A
        OR 58H
        LD H,A
        LD A,E
        AND 07
        OR A
        RRA
        RRA
```

Расчет адреса в файле атрибутов.

```
        RRA
        RRA
        ADD A,D
        LD L,A
        LD ADDR_A,HL      ;Запомнили адрес в файле
                          ;атрибутов.
        LD HL,ATRMAP      ;Адрес атрибутов в шаблоне.
        LD DE,ADDR_A      ;Адрес в файле атрибутов.
        LD C,SIZE_X       ;Счетчик знакомест по горизонтали.
LOOP_5  LD B,SIZE_Y       ;Счетчик знакомест по вертикали.
LOOP_4  LD A,(HL)
        INC HL
        LD (DE),A
        LD A,E            Переброска одного
        OR A              вертикального столбца.
        ADD A,20H
        LD E,A
        JR NC PASS
        INC D
PASS    DJNZ LOOP_4
        DEC C             ;Уменьшение счетчика столбцов.
        LD A,C
        OR B              ; Проверка регистра C на ноль.
        RET Z             ;Выход, если все столбцы закрашены
        LD DE,ADDR_A
        INC DE            Переход к следующему
        LD ADDR_A,DE      столбцу на экране.
        JR LOOP_5
```

2.3.3. Понятие об аркадных атрибутах экрана. Интерактивная графика.

Выше мы упомянули о том, что вместе с черно-белой информацией по шаблону может храниться и информация о его цветовых атрибутах, но можно хранить и так называемые аркадные атрибуты для каждого знакоместа.

Что такое аркадные атрибуты? Давайте вспомним всем известную программу "Pacman". Ваш герой живет и оперирует в некотором игровом пространстве. В одних местах он может спокойно перемещаться по экрану, другие части игрового поля ограничены стенами и недоступны для него. При встрече с "монстрами" он погибает, но иногда, если проглотит "таблетку силы", то наоборот - сам сможет уничтожать противников. Получается так, что разные участки экрана обладают разными свойствами. Стенки - непроницаемы, монетки - собираемы, таблетки, фрукты и т. п. - съедаемы, "монстры" - смертельны или уничтожаемы. Мы можем представить себе, что каждое знакоместо экрана обладает некоторым атрибутом или набором атрибутов. И если отдать на аркадные атрибуты экрана по одному байту на каждое знакоместо, то всего лишь 768 байтов оказывается достаточным, чтобы выразить огромное многообразие алгоритмов поведения персонажей.

Где хранить аркадные атрибуты? В ранних программах для "Спектрума" аркадные атрибуты совмещали с цветовыми. Это достаточно простое решение. Суть его состояла в следующем: программист принимал для себя некоторое соглашение типа:

Все, что окрашено в черный цвет - проходимо и проницаемо.
Все, что окрашено в зеленый цвет - съедаемо.
Все, что окрашено в красный цвет - смертельно.
Все, что окрашено в желтый цвет - собираемо.
Все, что окрашено в синий цвет - непроходимо.
Все, что окрашено в пурпурный цвет - непроходимо, но разрушаемо теми объектами, которые окрашены в голубой цвет (пулей, снарядом, мечом и т. п.).

Программа организована так, что прежде, чем выполнить печать героев и объектов на экране, проверяет, какой цветовой атрибут установлен в данном знакоместе (а как определить адрес в файле атрибутов по координатам знакоместа Вы уже знаете) и разрешает или запрещает эту печать. Она также может включить по результатам такой проверки процедуру, которая исполнит некоторое специфическое действие, например, начислит очки за удачно съеденный банан или изобразит сцену гибели "монстра", поражен-

ного лучом бластера и тоже начислит очки.

Вот мы с Вами и подошли к понятию "интерактивная графика". Как видите, в отличие от графики иллюстративной, которая служит как бы только приложением к программе, интерактивная графика активно влияет на ход ее работы. С одной стороны, подчиняясь командам от управляющего устройства (клавиатуры, джойстика) графические объекты печатаются на экране, стираются, печатаются в новом месте, а с другой стороны, при столкновениях (коллизиях) одних графических объектов с другими, включаются рабочие процедуры программы, которые изменяют ход ее работы. Так и получается, что практически вся графика в игровых программах - интерактивная.

"Цветовой" подход был хорош для ранних программ с очень простой псевдографикой. Для современных программ, отличающихся совершенной графикой и тщательно продуманными цветовыми решениями, "подвешивать" на цветовые атрибуты аркадные свойства было бы слишком грубым приемом. Экран выглядел бы слишком пестро. Возьмем программу "Sceptre of Bagdad". Веревка и колонна могут быть окрашены в одинаковые цвета, но по колонне нельзя забраться наверх, а по веревке можно. Это означает, что графический шаблон "веревка" имеет во всех своих знакоместах включенным аркадный атрибут "опора", а колонна - нет.

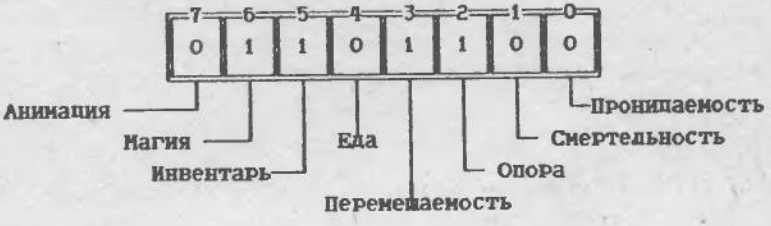
К счастью, проблема может быть решена достаточно просто. Мы можем создать в оперативной (не в экраной) памяти файл аркадных атрибутов, структура которого соответствовала бы структуре файла экранных атрибутов. Этот экран невидим, но он есть. Когда изображение Вашего героя печатается на экране вместе с цветовыми атрибутами, одновременно с этим на невидимом "аркадном" экране проверяются и "печатаются" в тех же координатах аркадные атрибуты Вашего героя или объекта.

Какие аркадные атрибуты могут быть выбраны? Это зависит исключительно от Вашей фантазии. Возьмем, например, ставший классическим "ботинок" из программы "Knight Lore". Мы не случайно останавливаемся на этой довольно старой программе, хотя

сейчас есть тысячи не менее достойных примеров. Дело в том, что этим "башмаком" фирма "ULTIMATE" произвела буквально революцию в игровом программном обеспечении и открыла принципиально новый жанр игр, к которому относятся столь любимые нашими читателями аркадные адвенТюры.

Какими аркадными атрибутами обладает этот старый ботинок?

1. Проницаемость = 0 (сквозь него нельзя пройти).
2. Смертельность = 0 (он не убивает).
3. "Опора" = 1 (на него можно встать или поставить на него другой объект).
4. Перемещаемость = 1 (его можно толкать и передвигать по игровому полю.
5. "Съедобность" = 0 (он несъедобен).
6. "Транспортабельность" = 1 (его можно взять и унести с собой, а потом выложить в другом месте). В этом смысле он является "инвентарем".
7. "Магия" = 1 (этот объект относится к числу тех, которые нужно отнести к чародею и бросить в волшебный котел).
8. "Анимация" = 0 (ботинок стоит неподвижно в отличие от некоторых других объектов, которые перемещаются).



Между делом может встать вопрос о том, достаточно ли разрешение 24 X 32 для аркадных атрибутов? Мы знаем, что для графической информации принято разрешение 192 X 256 и оно выглядит достаточным. Для цветовой информации разрешение 24 X 32 выглядит недостаточным, но приемлемым с большими натяжками. Для аркадных же атрибутов разрешение 24 X 32 оказывается более, чем достаточным. На практике вполне приемлемые результаты можно

получать с разрешением 6 X 8, т. е. когда по одному байту аркадных атрибутов выделено для каждого шаблона размером 4 X 4 знакоместа. Большинство же коммерческих программ используют разрешение 12 X 16, т. е. выделяя по байту аркадных атрибутов на каждый экранный блок размером 2 X 2 знакоместа.

Здесь же нужно отметить, что вовсе не обязательно укладывать все аркадные атрибуты в один байт. Вы можете выделить для этой цели два байта и хранить в одном аркадные атрибуты, а в другом - указание на адрес, в котором расположена процедура, обрабатывающая момент коллизии Вашего героя с данным шаблоном. Конечно, в одном байте разместить двухбайтный адрес невозможно, но можно, например, хранить только "смещение" в подготовленной Вами таблице переходов, а сам адрес перехода к нужной процедуре брать из этой таблицы.

В заключение укажем, что принципы интерактивной графики могут быть использованы не только в игровых программах, но и во вполне серьезных прикладных. Концепция "теневого" экрана специальных атрибутов может быть использована для организации на экране сложных графических меню, когда для каждого пункта такого меню по его координате можно найти в "теневом" экране атрибут, характеризующий этот пункт и содержащий указание на адрес перехода для исполняющей процедуры. По этому принципу несложно организовать и работу электронных таблиц, в которых при любом изменении значений чисел в клетках таблицы тут же переключается атрибут "теневого" экрана, указывающий на то, что значение в колонке K строки S было изменено, что запустит программу пересчета новых значений всей таблицы для данной строки и данного столбца.

2. 3. 4. "Печать" аркадных атрибутов.

Мы не случайно взяли слово "печать" в кавычки, ведь печатаются аркадные атрибуты на "теневом" экране, который находится не в экранной области памяти, а где угодно в оперативной. В то же время, процедура этой "печати" при правильном подходе может

практически ничем не отличаться от процедуры печати на основном экране, в чем Вы сейчас сможете убедиться. Считайте, что это тоже печать, хоть и не на экран, а в файл.

Представим себе, что у нас есть как бы три взаимосвязанных экрана.

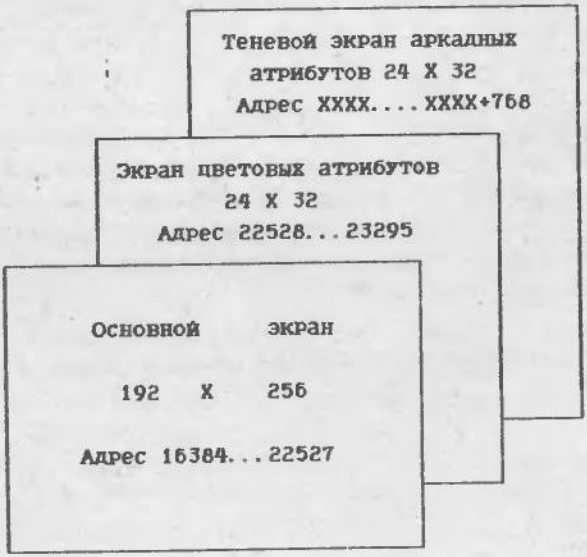


Рис. 5

Как всегда при работе с растровой графикой печать чего-либо на экране (в том числе и на "теневом") выполняется в два этапа. На первом этапе по координатам определяется адрес в файле, а на втором этапе выполняется копирование в этот адрес. Точно так же и сканирование по файлу для определения того, какой атрибут присвоен какому объекту тоже начинается с определения адреса в файле по координатам. Вспомним, как раскладывается адрес в файле экранных атрибутов по шестнадцати разрядам:

Рис. 6

Как видите, положение этого файла в оперативной памяти целиком определяется пятью старшими битами с 11-го по 15-ый. Измените хоть один из этих битов и Вы получите новый начальный адрес для "теневого" файла атрибутов. Так, если комбинация 0101 1000 дает адрес 22528, то например комбинация 1101 1000 уже даст начальный адрес 55296.

Вспомним процедуру для отыскания адреса в файле атрибутов по координате:

```
LD A,E
AND 18H
SRL A
SRL A
SRL A
OR 58H
LD H,A
LD A,E
AND 07
OR A
RRA
RRA
RRA
RRA
ADD A,D
LD L,A
```

Вот в этой точке и выставляются пять старших битов в порядке 01011, что определяет начальный адрес.

Замените эту операцию например на OR D8H и тогда начальный адрес файла будет равен 55296 и на экране он показан не будет, но его структура будет точно соответствовать структуре файла цветовых атрибутов.

Итак, процедура поиска адреса по координатам требует изме-

нения всего лишь в одном байте, а процедура печати атрибутов, приведённая выше в разделе 2.3.2. вообще никаких изменений не требует и мы ее не будем и рассматривать, коль скоро изменится только содержимое программной переменной ADDR_A.

Подходя к программированию творчески, можно вообще и не хранить в памяти компьютера процедуру поиска адреса в файле аркадных атрибутов, а использовать для этого процедуру поиска адреса в файле цветовых атрибутов, изменяя в ней один байт (например поставив D8 вместо 58) перед ее вызовом и восстанавливая его в прежнее состояние после ее работы.

2.4. Печать шаблонов без искажения фоновой картинки.

Приведенные выше процедуры впечатывают Ваш шаблон на экран и при этом "безвозвратно" затирают то изображение, которое под ним находилось. К сожалению, случаи, когда это так и должно быть, оказываются слишком редкими. Опять же, в рамки данной книги не входит рассматривать вопросы анимации и мультипликации, но сам процесс перемещения или изменения формы графических объектов на экране состоит ведь из трех этапов:

- печать спрайта (шаблона) на экране;
- стирание его же;
- печать этого спрайта в новом месте.

Если на первом этапе, при печати, не запомнить то, что было на экране в данном месте, то после стирания не удастся и восстановить исходную фоновую картинку.

В компьютерах некоторых систем, например "Коммодор-64", "Ямаха-MSX" есть специальные микросхемы - процессоры для работы со спрайтами. В "Спектруме" этого нет и Вам самим надо позаботиться, чтобы информация, хранившаяся на экране под Вашим напечатанным шаблоном не была утрачена.

Самое первое, что приходит в голову в такой ситуации - это взять изображение, хранившееся на экране до печати Вашего шаблона и скопировать в память в какой-то буфер для последующего восстановления. Может быть, это будет и весь экран или значительная его часть. Так, конечно, делают, хотя при этом есть и целый ряд неудобств. Это, во-первых, расход оперативной памяти на создание такого временного буфера. Во вторых, это приводит к снижению скорости работы программы, то есть та же мультипликация выполняется не самым плавным образом.

К счастью, у Вас есть элегантная возможность обойти эти проблемы, никакого изображения из экрана вовсе не запоминать и, в то же время, не утрачивать его при печати Вашего шаблона. Для этого вспомним логическую функцию XOR ("ИСКЛЮЧАЮЩЕЕ ИЛИ").

```
A       -   1010 0101
B       -   0011 1100
XOR B   -   1001 1001
```

Эта функция включает те биты, которые были включены в первом или во втором аргументе, но не в обоих вместе. Достоинство операции XOR B над содержимым аккумулятора состоит в том, что если мы еще раз повторим эту же команду, то получим в аккумуляторе то, что там было исходно. Проверьте:

```
A       -   1001 1001
B       -   0011 1100
XOR B   -   1010 0101
```

Итак, получается, что если мы хотим что-то напечатать на экране и не утратить хранившуюся там информацию, то нам надо не "затирать" байты экрана новыми, а сливать их командой XOR, а потом, когда надо будет стереть изображение с экрана, снова его же напечатать с командой XOR.

В этом смысле действие команды XOR в машинном коде эквивалентно действию команды OVER 1 при печати в БЕЙСИКе.

Как это сделать на практике? Очень просто. Возьмем центральный фрагмент процедуры, с помощью которой мы печатали шаблоны на экране (раздел 2.3.2.):

```
                ..........................................
LOOP_1  LD A,(HL)       ;Загрузили в аккумулятор
                        ;очередной байт шаблона.
        LD (DE),A       ;Перенос текущего байта из
                        ;оперативной памяти в экранную.
        INC D           ;Переход к следующей линии
                        ;знакоместа на экране снизу.
        INC HL          ;Переход к следующему байту
                        ;шаблона.
        DJNZ LOOP_1     ;Конец цикла копирования шаблона
                        ;8 X 8.
                ..........................................
```

Как видите, здесь командой LD (DE),A на экране производятся безвозвратные изменения. То, что было в экранном адресе, на который указывает регистровая пара DE, заменяется содержимым аккумулятора A (а это изображение нашего шаблона).

Попробуем внести в этот фрагмент необходимые изменения для того, чтобы шаблон не затирал изображение, а "сливался" с ним.

```
                ..........................................
LOOP_1  LD A,(DE)       ;Загрузили в аккумулятор
                        ;очередной байт экрана.
        XOR (HL)        ;Слияние по XOR с очередным байтом
                        ;шаблона.
        LD (DE),A       ;Печать байта на экран по XOR.
        INC D           ;Переход к следующей линии
                        ;знакоместа на экране снизу.
        INC HL          ;Переход к следующему байту
                        ;шаблона.
        DJNZ LOOP_1     ;Конец цикла копирования шаблона
                        ;8 X 8.
                ..........................................
```

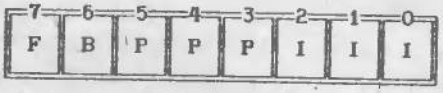
И теперь, когда надо будет "стереть" Ваш шаблон с экрана, Вы сможете вместо "стирания" напечатать его еще раз в том же самом месте той же самой процедурой.

2.4.1. Возможные проблемы цветовых атрибутов и борьба с "клэшингом".

Итак, мы с Вами договорились до того, что в принципе очень удобным, хотя и не единственным приемом печати шаблонов на экране, является "слияние" накладываемой и фоновой картинок с помощью логической функции XOR. Встает вопрос - а как при этом поступить с цветовыми атрибутами? Хотелось бы и их печатать с помощью XOR, дабы не приходилось сохранять в памяти то, что было на экране и потом восстанавливать. К тому же, мы хорошо помним, что в пределах одного знакоместа можно иметь только два цвета - INK и PAPER. Давайте рассмотрим возникающие при этом ситуации на серии конкретных примеров.

Предположим, Вы хотите решить вполне жизненную проблему - Вам надо "впечатать" на экран шаблон желтого автомобиля, а на экране в этом месте стоит зеленое дерево. То есть как бы Ваш желтый автомобиль едет на фоне зеленого леса.

Напомним конструкцию байта цветовых атрибутов экрана:

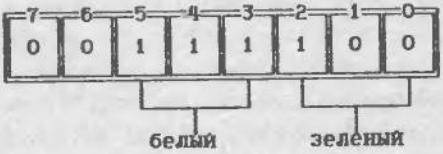


Здесь F - признак мигания (FLASH = 0,1);
B - признак яркости (BRIGHT = 0,1);
P - цвет фона (PAPER = 0...7);
I - цвет изображения (INK = 0...7).

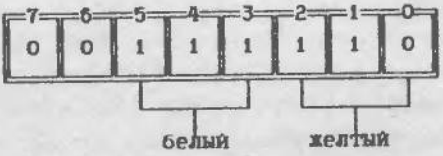
Для простоты будем считать, что наши изображения не имеют признаков FLASH и BRIGHT и они равны нулю.

Принцип 1.

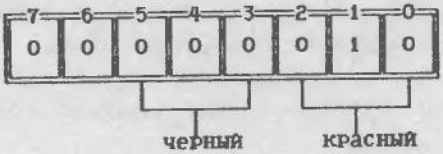
Если у нас лес нарисован зеленым по белому, то его атрибут будет равен:



А атрибут автомобиля, нарисованного желтым по белому:



После применения команды XOR мы получим:



Как видите, в результате и лес и автомобиль у нас станут после этого красными. Если с этим еще как-то можно смириться, то ух совсем никуда не годится та "черная дыра", которая возникнет в качестве фона для совмещенных изображений автомобиля и леса. Представляете, каково будет, если по красивому многоцветному экрану будет ездить такое черное пятно? Попробуйте изменить цвет PAPER у дерева и у автомобиля. Ставьте какие угодно сочетания и у Вас ничего путного не получится, пока Вы не выберете черный цвет в качестве фонового. Только он, поскольку его битовая конструкция равна 0 0 0, не превратится ни в какой другой, а останется тем, какой он есть.

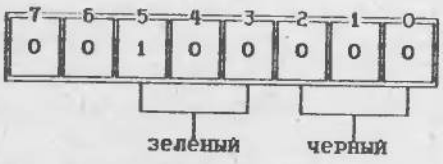
Итак, вывод первый: одним из использованных цветов при подготовке изображений должен быть черный цвет.

Принцип 2.

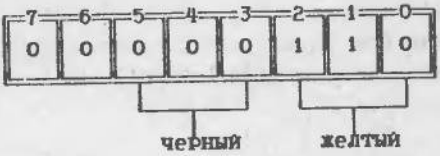
Хорошо, пусть один из цветов будет черным. Тем более, что в абсолютном большинстве программ так оно и есть и это не мешает им быть красивыми и многокрасочными. Встают только вопросы - как быть с тем, что у нас в результирующей картинке и лес и автомобиль окрасились в красный цвет, которого и в помине не было в нашей программе и какой цвет лучше назначить черным - INK или PAPER.

Какой цвет делать черным, мы еще разберем. А вот попробовать сделать черным и тот и другой - это интересно. Сделаем накладываемую картинку (автомобиль) желтым цветом INK на черном фоне PAPER, а вот фоновую картинку (лес) - наоборот - нарисуем зеленым цветом PAPER на черном цвете INK. Подход может быть несколько необычен, но именно так и делают. Именно рисуют цветом PAPER. Это означает, что в фоновой картинке там, где у Вас есть лес, там пикселы выключены, а там, где его нет - включены. Изображение получается инвертированным. Причем Вам вовсе не обязательно таким его рисовать в графическом редакторе. Вы можете его рисовать нормально - черным по белому, а потом инвертировать командой, которая есть в любом графическом редакторе и только после этого раскрашивать с помощью "пустой" кисти.

Посмотрим, что у нас получится на этот раз:



Авто:



XOR:

Как видите, в результате такого совмещения совместное изображение и леса и автомобиля станут желтыми, а фон для того и другого - зеленым. Это уже вполне приемлемый результат. С этим уже можно жить, хотя кое-что еще сделать конечно можно. Зеленая "дырка" фона в зеленом лесу смотрится совсем неплохо со всех точек зрения, а если Вы хотите, чтобы поменьше листвы окрасилось в желтый цвет, то рисуйте шаблон автомобиля так, чтобы его пикселы максимально перекрывали пикселы леса под ним. Например, по трассе движения автомобиля от леса оставьте только намек - отдельные зеленые пятна. Не забудьте также, что в результате наложения двух включенных пикселов по XOR результирующий пиксел выключается.

Итак, вывод второй: в накладываемом и фоновом изображениях черный цвет должен быть использован по-разному. Если у фона цвет PAPER - черный, то в накладываемом шаблоне черным принимаем цвет INK и, соответственно, наоборот.

Принцип 3.

Сейчас, когда мы разобрались с тем, как ведут себя цветовые атрибуты фоновой картинки и шаблона при печати на экране по

команде XOR, давайте вернемся к пиксельной (монохромной) информации и посмотрим, что происходит с изображениями автомобиля и леса, когда за черный цвет INK и PAPER у них приняты противоположные цвета.

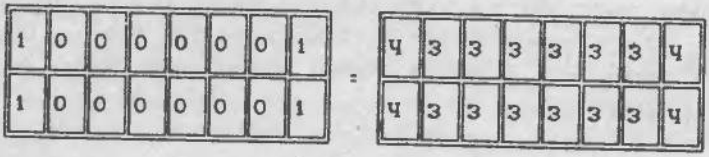
Мы не будем полностью изображать и лес и автомобиль, а рассмотрим у них для простоты по паре байтов. При этом окраску пикселов будем обозначать одной буквой:

З - зеленый; Ж - желтый; Ч - черный.

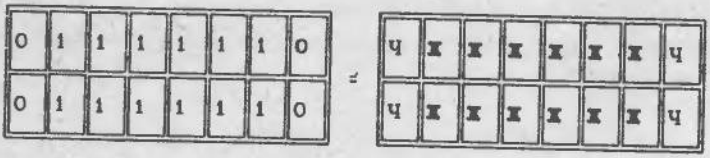
И рассмотрим два варианта:

Вариант 1: Для леса примем цвет INK - черным, а для автомобиля - наоборот черным будет цвет PAPER.

Лес

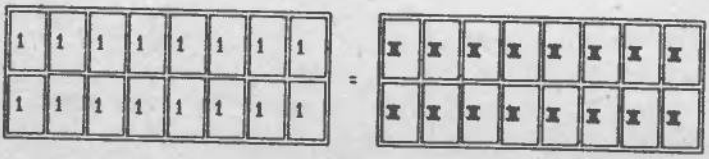


Автомобиль



В результирующем изображении включенные пикселы примут цвет INK, а он, как мы знаем, будет равен старому цвету INK автомобиля. Итак, мы получим:

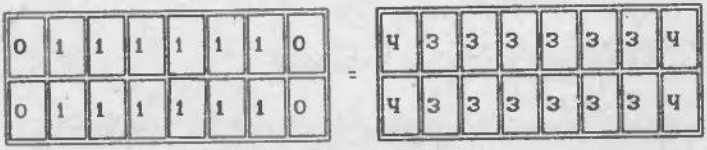
Результат



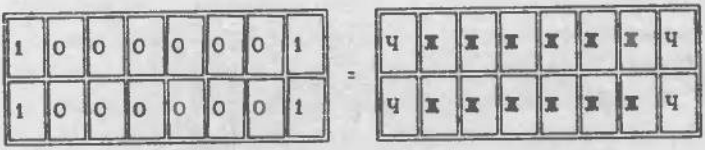
Как видите, преобладающим на экране будет желтый цвет. Создается впечатление, что автомобиль действительно едет по лесу и при этом он едет по переднему плану, закрывая деревья фона.

Вариант 2: Для леса примем цвет PAPER - черным, а для автомобиля - наоборот черным будет цвет INK.

Лес

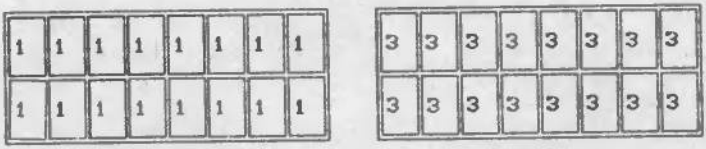


Автомобиль



В результирующем изображении включенные пикселы примут цвет INK, а он, на этот раз будет равен цвету INK леса. Итак, мы получим:

Результат:

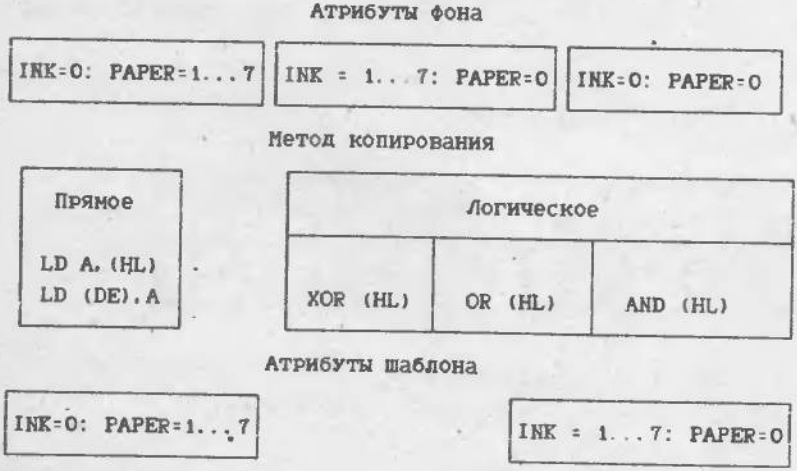


Как видите, преобладающим на экране будет зеленый цвет, т. е. автомобиль едет за деревьями, на заднем плане.

Итак, вывод третий: при наложении одного изображения на другое по XOR, черный цвет PAPER следует принимать для того изображения, которое должно занять передний план. Для изображений заднего плана наоборот за черный принимаем цвет INK. Иначе говоря: фоновое изображение подготавливаем с помощью графического редактора в инвертированном виде - рисуем цветом PAPER.

2.5. Трехмерная растровая графика.

Мы рассмотрели широкие возможности применения копирования растровой и цветовой составляющих с помощью команды XOR. Безусловно, этот метод позволяет достичь важных результатов, но наверное этот метод не единственный. И действительно, существует ведь еще копирование с помощью логических команд AND (И) и OR (ИЛИ). Надо только в процедуре, приведенной в разделе 2.4 заменить команду XOR (HL) на AND (HL) или на OR (HL). Сколько же всего возможно различных сочетаний при копировании растровой графики Вы можете увидеть на нижеприведенной схеме.



Всего возможны 3*4*2 = 24 сочетания. Конечно, такое многообразие нам никак не рассмотреть на страницах данной книги, но в этом, по-видимому, и нет необходимости, ведь главное - указать на приемы и подходы, а дальше читатель сам может развивать свое мастерство и изобретать новые способы. А мы же сейчас остановимся на одной из областей растровой графики, в которой нашла свое применение другая логическая функция - OR. Речь пойдет о трехмерной растровой графике и мы опять вспомним программу "Knight Lore" фирмы ULTIMATE, хотя вместо нее можно было бы указать любую из многих сотен программ, в которых использован аналогичный прием.

Взгляните на рисунок 7. Что Вам напоминает это изображение? Вряд ли оно похоже на что-то более менее содержательное.

Рис. 7

А если его же напечатать на черном фоне экрана, как показано на рис. 8? Совсем другое дело, не правда ли?

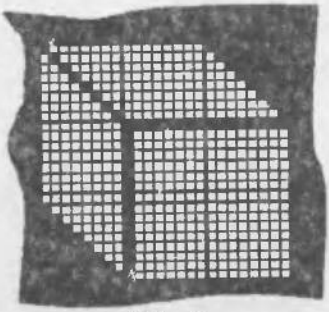


Рис. 8

Да, это тот самый элементарный блок, из которых построен замок злого волшебника. Блок, как видите, трехмерный, но пусть Вас это не тревожит - шаблон блока (рис. 9) самый обычный размером 3 X 3 знакоместа. Многое можно изобразить на экране с помощью таких очень простых шаблонов.

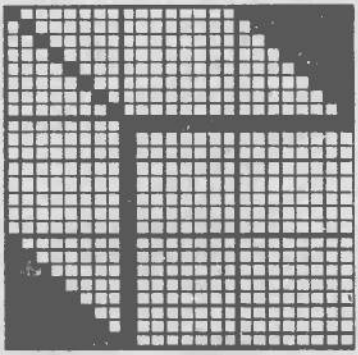


Рис. 9

Но, коли уж мы заговорили об имитации трехмерного игрового пространства, нам надо рассмотреть и то, как из таких элементарных блоков строится трехмерная конструкция. Все очень просто - эти блоки должны печататься со смещением. На рис. 10 показана простейшая конструкция из трех блоков (A, B, C).

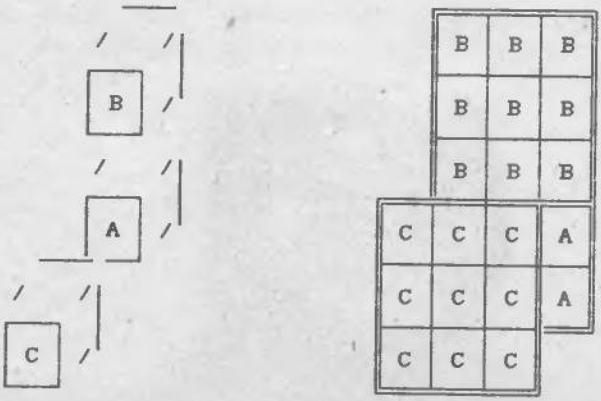


Рис. 10

При таком наложении шаблона на шаблон произойдет конфликт между пиксельной конструкцией этих шаблонов. Посмотрите на Рис. 11. что происходит. если накладывать их друг на друга простым копированием.

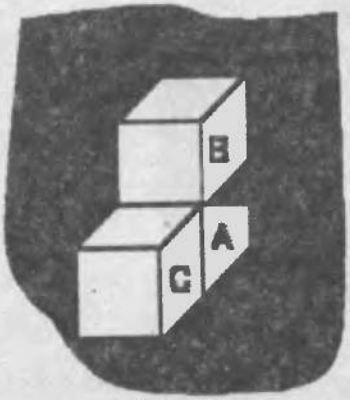


Рис. 11

Проблему представляют черные углы по диагонали шаблона. При наложении черных выключенных пикселов на белые включенные можно получить белые только если пользоваться наложением по XOR или по OR. т. к. только для этих функций

1 OR 0 = 1

1 XOR 0 = 1

Вместе с тем, нам не удается воспользоваться и командой XOR. т. к. при этом наложение белого на белое даст черное пятно.

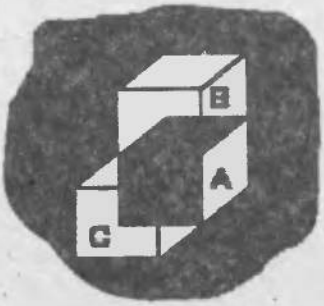


Рис. 12

В этом случае нас выручает команда OR. Как видите, при таком наложении все в порядке (Рис. 13).

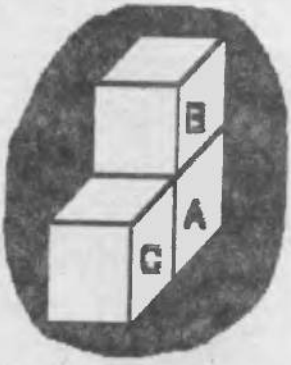


Рис. 13

Конечно, кое-что при таком подходе мы и теряем. Например, мы теряем возможность использовать печать того же самого шаблона еще раз, чтобы выполнить стирание без искажения содержимого фона. Но это для нас и не страшно, ведь по сути дела здесь фоном является просто чистый черный экран, а когда мы печатали один блок поверх другого, мы печатали фактически один шаблон поверх другого шаблона, а не поверх фона. Восстановить простой чистый черный фон очень просто, напечатав пустой шаблон размером 3 X 3 знакоместа, что не труднее, чем повторять печать того же самого шаблона.

У нас могут быть проблемы с цветовыми атрибутами. Но, простите, а где в игровых программах с трехмерной графикой применяют более двух цветов? Если это и делают, то только по полям игрового поля, а не на основном игровом экране.

Итак, мы пришли к еще одному основополагающему выводу:

> Вывод четвертый: Если мы накладываем шаблон на фоновую картинку, то лучше пользоваться командой XOR. Если мы накладываем шаблон на шаблон, то лучше пользоваться командой OR, фоновую картинку упростить до минимума, а от применения палитры цветов воздержаться.

2.6. Упаковка экранной информации.

Здесь под словом "упаковка" мы понимаем не компрессию изображения по принципу выявления и устранения последовательностей повторяющихся байтов. Речь идет о том, что если в программе сотни экранов, то нужны какие-то другие приемы их упаковки, например создание "карты" для каждого экрана. Рассмотрим рис. 14.

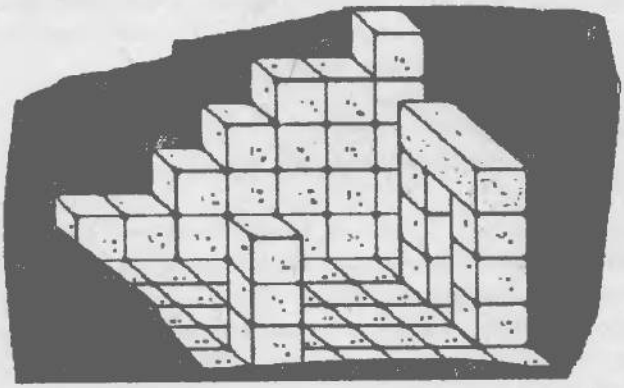


Рис. 14

Здесь показана конструкция одной типовой комнаты средневекового замка. Все содержимое этой комнаты в трехмерном пространстве можно представить как бы состоящим из элементарных кубических блоков размером 2 X 2 X 2 знакоместа каждый (Рис. 9). Эти блоки на плоскость проецируются в виде плоских шаблонов размером 3 X 3 знакоместа. Конечно же при создании комнаты могут быть использованы блоки самого разного вида и самых разных свойств (например дверные блоки служат для прохода в другие комнаты). И нам выгоднее хранить не картинку каждой комнаты, а ее раскладку на блоки с указанием того, какой блок где находится.

В этом случае всю нашу комнату можно представить, например, как набор из 9 X 7 X 5 блоков, т е. в виде последовательности длиной всего 315 байт. В каждом байте стоит номер блока, который здесь должен быть поставлен. В тех местах (а их боль-

шинство), где нет вообще ничего, можно поставить ноль. Разумеется, большинство блоков повторяется и если теперь скомпрессировать карту данной комнаты, то окажется, что на ее хранение достаточно 15 - 20 байтов и теперь ее можно заархивировать вместе с другими комнатами в единый длинный файл, из которого можно по номеру комнаты в любой момент извлечь ее "карту", декомпрессировать и напечатать. Вот так и получается, что в программе "Head over Heels" программистам удалось заложить более 300 очень непростых экранов, принадлежащих к разным замкам, всего лишь в 5 с небольшим килобайт информации.

2.7. Архивация данных.

Архивация данных - это не просто компрессия, это еще и объединение скомпрессированных изображений в единый архив (библиотеку) в заданном формате, чтобы потом дезархивирующая процедура могла найти нужное изображение, разархивировать его и передать на экран.

Конечно, архивировать можно не толко шаблоны изображений, но и карты комнат и текстовые сообщения и экраны и многие другие данные. Разные объекты могут, в принципе, требовать разного подхода, но мы остановимся на одном примере и рассмотрим весь цикл подготовки библиотеки шаблонов для будущей программы.

Этот цикл состоит из следующих этапов:

1. Подготовка шаблонов изображений с помощью графического редактора или другой доступной программы и отгрузка их на ленту.

2. Поочередная загрузка шаблонов, компрессирование их и выгрузка на ленту в виде единого файла в заданном формате.

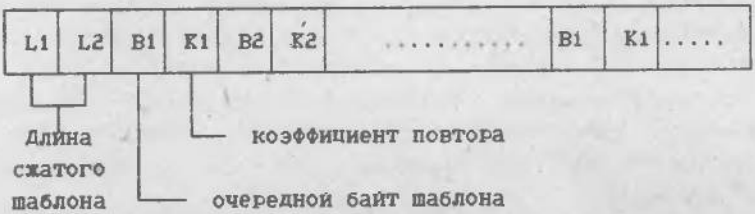
3. Подготовка дезархивирующей процедуры для включения ее в состав будущей программы.

Рассмотрим требования к программам архивации и дезархивации. К архивирующей процедуре никаких особых требовании нет. Нам не очень важно ее быстродействие и нет никаких проблем с объемом занимаемой памяти. Архивация данных является просто техническим приемом, которым Вы можете заниматься в удобное Вам время не спеша. Единственное требование к компрессирующей процедуре - это то, что желательно иметь на выходе код как можно меньшего размера, если это не слишком скажется на быстродействии дезархивирующей процедуры, ведь можно сделать такое многоступенчатое компрессирование, при котором дезархивирующая процедура будет работать слишком медленно. Но так или иначе, а выбор во всех случаях за Вами и никому, кроме Вас лучше не известны требования, которые Вы предъявляете к своей будущей программе и Вы сами определитесь, в какой степени Вам нужно быстродействие и насколько напряженно обстоит дело с объемом доступной памяти.

К дезархивирующей процедуре предъявляются очень жесткие требования. Она работает в составе Вашей программы и потому должна и занимать минимальный объем памяти и работать как можно быстрее. Конечно, Вы понимаете, что это зависит от алгоритма, по которому она работает и, тем самым, связано с принятым форматом данных и с методом компрессии.

2.7.1. Методика компрессии.

Принципов компрессии данных много. Для текстовых данных лучше подходят одни приемы, для числовых - другие. Графические данные характеризуются высокой степенью повторяемости одинаковых байтов и потому здесь наиболее широко применяется прием, когда компрессирующая программа проверяет количество таких повторов и после каждого байта, принятого из источника, ставит число, указывающее на количество повторений, например:



Отводить первые два байта на указание длины сжатого шаблона нам необходимо для того, чтобы при дезархивации программа могла найти начало нужного шаблона по его номеру.

В книге "Элементарная графика" мы рассмотрели несложную процедуру для компрессирования экранов. Здесь можно было бы воспользоваться ею же, но, к сожалению, есть существенные отличия. Дело в том, что там нам хорошо известны были и начало и конец исходного файла (это экранный файл) здесь же их надо вводить и хранить. Поэтому регистров общего назначения BC, DE и HL нам уже не хватает, приходится привлекать к работе индексные регистры процессора - IX и IY. Впрочем, это ведь только пример и Вы сможете сами переделать процедуру так, как Вам будет удобно.

Для создания процедуры компрессии файла шаблонов решим для себя, что регистровая пара IX у нас будет указывать на байт источника, регистровая пара IY - на байт приемника, регистровая пара BC будет служить счетчиком просмотренных байтов источника, а регистр L будет служить счетчиком коэффициента повтора одинаковых байтов. При этом здесь возможны два варианта. Иногда при компрессии экранов целесообразно подсчитывать коэффициент повтора в пределах до 65535, т.е. под него выделяют не регистр, а регистровую пару и в сжатом файле под каждый коэффициент отводятся по два байта. Такой подход дает ощутимый выигрыш, когда речь идет о компрессии экрана целиком, да к тому же если он не очень заполнен информацией. В нашем случае, когда речь идет только о компрессии шаблонов, лучше под счетчик повторов выделять только один регистр, т.к. размеры шаблонов обычно невелики и здесь практически не встретятся коэффициенты повторов больше, чем 255. Регистр A используем для хранения текущего байта, принятого из источника, а регистр E для хранения сравниваемого с ним очередного байта с целью выявления повтора.

Для нашей процедуры нам потребуются программные переменные
BEGIN - адрес начала файла шаблонов в памяти (адрес источника);
STORE - адрес начала скомпрессированного файла шаблонов (адрес приемника).

Тогда процедура компрессии файла шаблонов может выглядеть, например так:

```
        PUSH IX          ;Сохранение регистров на стеке.
        PUSH IY          ;
        LD IX, BEGIN     ;Адрес начала файла шаблонов.
        LD BC, (BEGIN)   ;Длина шаблона.
        INC IX
        INC IX
        DEC BC           ;Уменьшили длину источника на
        DEC BC           ;два байта (в них хранилась сама
                         ;эта длина).
        LD IY, STORE     ;Адрес начала скомпрессированного
                         ;шаблона.
        INC IY           ;Пропускаем два байта для того,
        INC IY           ;чтобы впоследствии в них
                         ;подставить длину скомпрессирован-
                         ;ного шаблона.
RETURN  LD L, 01         ;Инициализация счетчика повторов.
        LD A, (IX+0)     ;Приняли текущий байт.
AGAIN   INC IX           ;И перешли к следующему.
        DEC BC           ;Уменьшаем счетчик источника.
        PUSH AF          ;Освобождаем регистр A и выполняем
        LD A, C          ;проверку на конец источника.
        OR B
        JR Z, END        ;Если весь источник просмотрен, то
                         ;выход через метку END.
        POP AF           ;Восстановление регистра A.
        LD E, (IX+0)     ;Очередной байт приняли в
                         ;регистр E.
        CP E             ;И сравнили его с предыдущим в A.
        JR NZ, PASS      ;Если нет повтора, то обход на
                         ;PASS.
        INC L            ;Если есть повтор, то наращиваем
                         ;счетчик повторов L.
        JR NZ, AGAIN     ;И переходим для анализа
                         ;следующего байта источника.
```

```
        DEC L           ;Если счетчик обнулился, т.е.
                        ;переполнился, уменьшаем его на
                        ;единицу и перестаем нарашивать.
                        ;В качестве коэффициента повтора
                        ;будет взято число 255.
PASS    LD (IY+0),A     ;Перенесли байт из источника
                        ;в приемник.
        INC IY          ;Указатель в приемнике переставили
                        ;вверх.
        LD A,L          ;Ввели коэффициент повтора.
        LD (IY+0),A     ;Отправили его в приемник.
        INC IY          ;Указатель приемника - вверх.
        JR RETURN       ;Возврат для обработки очередного
                        ;байта.

        END     POP AF  ;Первым делом надо восстановить
                        ;баланс на стеке, т.к. выход на
                        ;END происходил после команды
                        ;PUSH AF, для которой не было
                        ;соответствующей команды POP AF.
        LD (IY+0),A     ;Последний байт выдается в
                        ;приемник.
        INC IY          ;Туда же выдается его
        LD A,L          ;коэффициент повтора
        LD (IY+0),A     ;из регистра L.
        PUSH IY         ;Аналогично операции LD HL,IY,
        POP HL          ;которой нет в системе команд
                        ;Z-80.
        LD DE,STORE
        LD (STORE),HL   ;Изменили программную переменную
                        ;STORE, чтобы подготовить приемник
                        ;для нового шаблона.
        AND A           ;Сброс флага переноса для того,
                        ;чтобы он не влиял на следующую
                        ;операцию SBC, т.к. операции SUB
                        ;для регистровых пар в системе
                        ;команд Z-80 нет.
        SBC HL,DE       ;Определили длину скомпрессиро-
```

```
                    ;ванного шаблона.
LD IY,DE
LD (IY+0),L         ;Сохранили длину шаблона вместе
LD (IY+1),H         ;с самим сжатым шаблоном.
POP IY              ;Восстановление ранее сохраненных
                    ;регистров перед окончательным
POP IX              ;выходом из процедуры.
RET
```

2.7.2. Буферизация экрана.

Декомпрессия шаблонов может производиться как непосредственно на экран, так и в заранее организованный для этой цели буфер. Возможна и двойная буферизация, когда каждый шаблон сначала декомпрессируется в буфер шаблонов, а потом оттуда впечатывается в буферный экран, который в свою очередь потом копируется в экранную область памяти.

Вроде бы при таком порядке действий значительно замедлится быстродействие программы, но это не совсем так. Программисту ведь не нужна скорость ради скорости. Ему нужна скорость работы только для того, чтобы освежение (перестроение) экрана происходило бы как можно более быстро, а самые быстрые результаты показывает применение копирования экрана с помощью команды блочной переброски байтов LDIR (справедливости ради надо отметить, что есть и еще более быстрые способы переброски больших блоков данных и об этом мы писали на страницах нашего "фирменного" издания "ZX-РЕВЮ").

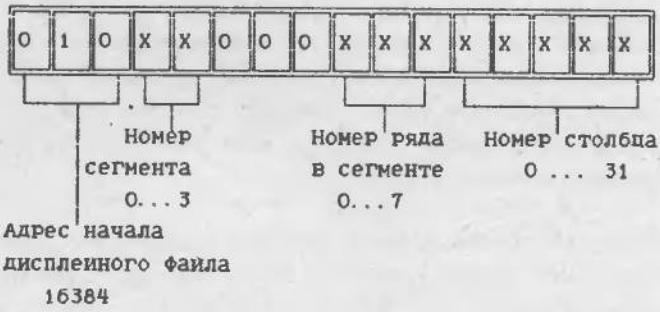
```
LD HL,BEGIN     ;Начало буферного экрана.
LD DE,4000H     ;Начало экранной области.
LD BC,1B00H     ;Длина дисплейного файла вместе
                 с цветовыми атрибутами.
LDIR            ;Переброска.
```

Поэтому заполнение буферного экрана можно производить в тот момент, когда программа находится в состоянии ожидания действий

пользователя, а это, кстати, основное состояние для любой программы. То ли пользователь читает какое-то сообщение, то ли думает, куда бы ему пойти дальше, а в это время незаметно для него идет "печать" в невидимый буферный экран.

"Печать" в буферный экран может производиться теми же самыми процедурами, которыми мы печатаем в основной экран с минимальными доработками. Напомним, как связаны адрес в экранной памяти с координатами позиции печати:

A: Координаты заданы в знакоместах.



Как видите, местоположение экранного файла в оперативной памяти определяют три старших бита, в которых стоит число 0 1 0. Устанавливаются эти биты командой OR 40H. Ниже приведен фрагмент процедуры (см. раздел 1.3), которая это делает

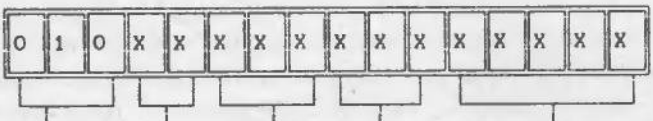
```
LD A,E
AND 18H
OR 40H
.......
```

Если эту раскладку изменить, то соответственно Вы получите новые адреса, в которые сможете "печатать" свои шаблоны, как на экран.

0 1 0 - адрес 16384 - OR 40H

0 1 1	-	адрес 24576	-	OR 60H
1 0 0	-	адрес 32768	-	OR 80H
1 0 1	-	адрес 40960	-	OR A0H
1 1 0	-	адрес 49152	-	OR C0H
1 1 1	-	адрес 57344	-	OR E0H

В: Координаты заданы в пикселах.



Здесь точно так же три старших бита, имеющие раскладку 010, определяют адрес 16384. Но устанавливаются они не так просто, как это было в предыдущем случае. Ниже мы привели фрагмент процедуры из раздела 1.3, в котором выполняется эта установка.

```
.....
LD E,A
AND A          ------------ 0 --+
RRA                             |
SCF            ------------ 1 --+  Этими тремя командами
RRA                             |  выставляется конфигу-
AND A          ------------ 0 --+  рация 010.
RRA                                Ее можно изменить и
XOR E                              сделать другую конфигу-
AND 0F8H                           рацию.
XOR E                              Там, где Вам нужен ноль,
LD H,A                             нужно поставить команду
.....                              AND A, а там, где нужна
                                   единица, нужно поставить
                                   команду SCF
```

Таким образом, если Вы хотите, например, чтобы буферный экран располагался начиная с адреса 57344, Вам надо заменить эти команды на:

```
.....
SCF  ———————— 1
RRA
SCF  ———————— 1
RRA
SCF  ———————— 1
RRA
....
```

Примечание: если координаты заданы в пикселах, то мы копируем шаблон на экран с помощью процедуры PLOT, расположенной в ПЗУ по адресу 22E5H = 8933 DEC (см. 2.2.2). Но если мы хотим "печатать" не на основной экран, а в "теневой", то пользоваться этой процедурой нельзя, ведь надо внести изменения, а в ПЗУ их внести нельзя. Тогда сами напишите свою процедуру, эквивалентную PLOT (мы рассмотрели ее в книге "Элементарная графика"), и внесите необходимые изменения.

2.7.3. Методика декомпрессии.

Для простоты изложения мы предположим, что декомпрессия производится в буфер, откуда шаблон может быть напечатан с помощью рассмотренных выше процедур как в дисплейный файл, так и в буферный экран. Мы также не будем останавливаться на простой рассмотренной выше проблеме поиска шаблона по его номеру и сразу примем, что регистровая пара DE указывает на начало компрессированного шаблона, а в регистровой паре HL устанавливаем адрес буфера, в который происходит декомпрессия.

```
BUFFER    DEFW  0000
STORE     DEFW  0000

          LD DE,STORE      ;Адрес начала скомпрессированного
                           ;шаблона.
          LD L,(DE)        ;Младший байт длины шаблона
          INC DE           ;Переход к старшему байту длины
          LD H,(DE)        ;Старший байт длины шаблона.
          DEC DE           ;Вернулись к началу шаблона.
```

```
        ADD HL,DE       ;И определили адрес его конца.
        PUSH HL         ;Сохранили на стеке адрес конца
                        ;компрессированного шаблона.
        INC DE          ;Пропуск двух байтов, хранящих
        INC DE          ;длину шаблона.
        LD HL,BUFFER    ;Адрес буфера-приемника.
AGAIN   LD A,(DE)       ;Приняли байт шаблона.
        PUSH AF         ;Запомнили его на стеке.
        INC DE          ;Переход к новому байту.
        LD A,(DE)       ;Приняли очередной байт
                        ;(это коэффициент повтора)
        INC DE          ;Переход к новому байту.
        LD B,A          ;В регистре B организуется
                        ;счетчик повторов.
        POP AF          ;Восстановили байт шаблона.
LOOP    LD (HL),A       ;И поместили его в буфер.
        INC HL          ;Следующий байт буфера.
        DJNZ LOOP       ;Повторяем цикл от LOOP столько
                        ;раз, каков коэффициент повтора.
        POP BC          ;Восстановили адрес конца шаблона.
        LD A,D          ;
        CP B            ;  Проверка на конец шаблона.
        JR NZ,PASS      ;
        LD A,E          ;
        CP C            ;
        JR NZ,PASS      ;
        RET             ; Если конец, то выход.
PASS    PUSH BC         ;Если не конец, то опять запомнили
                        ;адрес конца шаблона на стеке.
        JR AGAIN        ;И повторяем процесс декомпрессии.
```

2.8. Печать нестандартными шрифтами.

Воспользовавшись тем, что в данной главе идет речь о растровой графике и о печати на экране растровых шаблонов, мы не могли не остановиться и на вопросе о возможности печати растровыми шрифтами, отличающимися от стандартно принятых 8 X 8 пик-

селов. Вопрос этот относится к очень важным, хотя и малоосвещенным. Можно сказать так, что применение нестандартных шрифтов в игровых программах имеет более широкий характер, чем освещение этого вопроса в мировой литературе.

А между тем, коль скоро в предыдущих разделах мы освоили печать на экране заранее подготовленных шаблонов произвольного размера, то нам теперь рукой подать до того, чтобы изготовить шаблоны нестандартных символов и подойти к нестандартным растровым шрифтам. Более того, мы пойдем дальше и попробуем реализовать такую печать даже из БЕЙСИКа с помощью обычного оператора PRINT#. И такая возможность в "Спектруме" предусмотрена. Ее можно реализовать, организовав новый канал для печати.

Продемонстрируем это на практическом примере. Зададимся целью спроектировать нестандартную процедуру печати, которая позволяла бы выводить на экран текст, состоящий из символов размером, например, 5*5 пикселов. Такая процедура плотной печати может вполне найти практическое применение в программах, в которых необходимо в пределах одного экрана разместить максимально возможное количество информации. Это может быть, например, программа-тест, выдающая результат по завершении тестирования.

2.5.1. Шрифты размером менее, чем 8X8.

Условимся, что один символ будет иметь размеры в ширину и в высоту по 5 пикселов. В стандартном символьном наборе для хранения шаблона одного символа используются 8 байтов, по одному на каждую пиксельную линию. Причем первый и последний байты, как правило, равны нулю, этим обеспечивается интервал по вертикали между строками текста. Условимся, что наш шаблон одного символа будет занимать не более 5 пиксельных линий. А для того, чтобы обеспечить расстояние между строками по вертикали, предусмотрим при печати одну пустую пиксельную линию.

Учитывая предложенные условия, рассчитаем количество выводимой на один экран текстовой информации. В одной строке может

быть выведено: 256/5=51 символ. По вертикали может быть размещено: 176/6=29 строк (одна строка - символы по 5 пикселов в высоту плюс 1 пустая пиксельная линия).

Координаты для вывода символов на экран удобнее задавать не в пикселах, а в знакоместах, аналогично AT Y,X. Но мы, чтобы значительно не усложнять процедуру печати, координаты будем задавать не при помощи AT, а при помощи двух системных ячеек, занося туда значения перед началом печати. В процессе печати текста значения этих ячеек будем программно менять для того, чтобы отслеживать текущую позицию печати. Кроме того, опять же для упрощения процедуры печати, условимся, что по достижении конца экрана, вывод будет прекращаться. Проверка на необходимость скроллинга и сам скроллинг - достаточно емкая часть процедуры печати, поэтому от них откажемся, уделив особое внимание тому, как же организовать вывод текста на экран в 29 строк по 51 символу в строке.

Теперь несколько слов надо сказать о символьном наборе. Можно, конечно, использовать символьный набор в стандартном формате 8*8 пикселов, задав его таким образом, что символы будут занимать 5*5 пикселов. Тогда три байта из восьми будут пустой тратой памяти. Можно сократить эти потери, сформировав символьный набор так, что на каждый символ будет отведено по 5 байтов. Тогда полный символьный набор для 96 символов будет иметь объем всего 96*5=480 байтов вместо 768 для стандартного символьного набора. Можно, в принципе, пойти еще дальше, и сократить недоиспользуемые 3 бита в каждом байте, однако, видимо это уже нецелесообразно, так как полученная дополнительная экономия (всего-то примерно полторы сотни байтов) частично будет "съедена" усложнившейся процедурой декодирования, да еще, к тому же, из-за этого несколько замедлится работа процедуры печати. Логичнее, видимо, ограничиться "упаковкой" символьного набора в формат 5*8 (по 5 байт на символ).

Вручную, конечно, выполнить такую работу трудно, поэтому в комплект процедуры плотной печати входят две небольшие программы-утилиты. Во-первых, это "упаковщик" символьного набора из

формата 8*8 в формат 5*8. Он выбрасывает "лишние" три байта каждого символа и пододвигает следующий символ к предыдущему. Во-вторых, это "распаковщик", который выполняет противоположную задачу: "растягивает" упакованный символьный набор до стандартного формата 8*8 для того, чтобы можно было вносить изменения в символьный набор при помощи стандартного программного обеспечения, например, "ARTSTUDIO".

Формирование символьного набора иллюстрирует рисунок 15. Здесь показан результат работы с русско-латинским символьным набором (ASCII, комплект "НС" КОИ-7) в графическом редакторе "ARTSTUDIO". Обратите внимание: все символы "прижаты" к верхнему левому углу поля 8*8 пикселов, занимая, таким образом, только первые 5 байтов из восьми. Ширина символов, по-возможности, занимает 4 пиксела, однако, там, где это невозможно, например для букв "Ш", "Ж", "W" и др. - все пять пикселов. При этом, правда, не будет разделения с соседним символом. Три вертикальных столбца справа у каждого символа (три младших бита в каждом байте) вообще не используются.

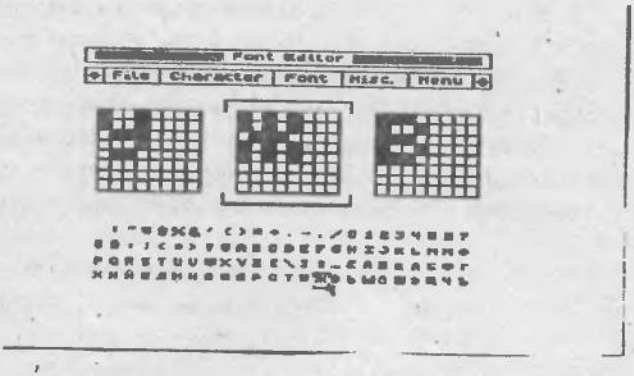


Рис. 15

Полная процедура печати приведена в листинге 2.8.1 Мы расположили ее с адреса AB00H (43776). Утилита-"упаковщик" символьного набора расположена с адреса AB80H (43904), а утилита-"распаковщик" - с адреса ABB8H (43960). Весь блок кодов проце-

дуры плотной печати оформлен в файл "#4"CODE 43776,256. (Определим для плотной печати поток #4.) Упакованный символьный набор расположен сразу же за процедурой печати с адреса AC00H (44032), и может быть включен в этот же файл. Значение CHARS для этого символьного набора будет равно: 44032-32*5=43872 (AB60H).

На входе в процедуру печати в аккумуляторе должен находиться код печатаемого символа, а в системных ячейках ABFEH (44030) и ABFFH (44031) должны быть заданы начальные координаты печати, соответственно: колонка (0...50) и строка (0...28).

Листинг 2.8.1.

```
AB00  FE20     CP   #20      ;Проверка, не является ли печата-
                              ;емый код управляющим символом.
AB02  D8       RET  C       ;Если да, то возврат из процедуры
                              ;печати.
AB03  FE80     CP   #80      ;Проверка, не является ли печатае-
                              ;мый код токеном ключевого слова
                              ;или символом блочной графики.
AB05  D0       RET  NC      ;Если да, то возврат из процедуры.
```

Следующая часть программы вычисляет координаты в пикселах места для печати символа по заданным координатам в знакоместах. Последние заданы при помощи POKE в ячейках ABFEH (координата X) и ABFFH (координата Y). Результат рассчета будет получен в регистре BC.

```
AB06  F5       PUSH AF      ;Сохранение на стеке кода печатае-
                              ;мого символа.
AB07  21FFAB   LD   HL,#ABFF;В HL адрес ячейки, в которой на-
                              ;ходится номер строки печати.
AB0A  7E       LD   A,(HL)  ;Значение, содержащееся там, срав-
AB0B  FE1D     CP   #1D      ;нивается с максимально
                              ;допустимым (1DH=29 строк).
AB0D  D29F1E   JP   NC,#1E9F;Если это значение превышено, то
                              ;переход на процедуру REPORT B в
```

```
                                    ;ПЗУ для вывода сообщения об
                                    ;ошибке "Integer out of range".
AB10  2B        DEC   HL            ;Теперь в HL адрес ячейки, в ко-
                                    ;торой находится значение колонки
                                    ;печати.
AB11  7E        LD    A,(HL)        ;Значение, содержащееся там срав-
AB12  FE33      CP    #33           ;нивается с максимально допустимым
                                    ;(33H=51 символ в строке).
AB14  3806      JR    C,#AB1C       ;Если результат не превышен, то
                                    ;переход на продолжение процедуры
                                    ;печати.
AB16  AF        XOR   A             ;Иначе - переход на новую строку.
AB17  77        LD    (HL),A        ;Для этого обнуляется счетчик
AB18  23        INC   HL            ;колонок, а счетчик строк
AB19  34        INC   (HL)          ;увеличивается на единицу.
AB1A  18EE      JR    #AB0A         ;Возврат на контроль счетчика
                                    ;строк, не превышен ли предел.
AB1C  4F        LD    C,A           ;К этому моменту в аккумуляторе
                                    ;находится значение координаты X
                                    ;(в знакоместах). Теперь оно и в
                                    ;регистре C.
AB1D  87        ADD   A,A           ;Таким способом производится
AB1E  87        ADD   A,A           ;умножение этого значения на 5
AB1F  81        ADD   A,C           ;для рассчета координаты в
                                    ;пикселах.
AB20  4F        LD    C,A           ;Полученная координата X в пикс-
                                    ;лах сохраняется в регистре C.
AB21  23        INC   HL            ;Переходим к адресу, где задан
                                    ;номер строки печати.
AB22  7E        LD    A,(HL)        ;Приняли это значение.
AB23  47        LD    B,A           ;Таким способом производится
AB24  87        ADD   A,A           ;умножение  на 6 для расчета
AB25  80        ADD   A,B           ;координаты в пикселах
AB26  87        ADD   A,A           ;т.к. шаг строк - 6 пикселов.
AB27  47        LD    B,A           ;Значение Y в регистре B.
AB28  3EAE      LD    A,#AE         ;Его надо вычесть из числа 174,
AB2A  90        SUB   B             ;т.к. отсчет координаты в пикселах
                                    ;ведется из левого нижнего угла.
```

Максимальное значение координаты Y равно 175, однако выбрана величина 174, т.к. тогда пустая пиксельная линия будет отделять текст от бордюра, который может иметь тот же цвет, что и текст. Таким образом предотвращается сливание текста с бордюром.

```
AB2B  47       LD   B,A       ;Скорректированная координата Y
                                ;сохраняется в регистре B.
AB2C  F1       POP  AF        ;Восстановление кода печатаемого
                                ;символа.
```

Теперь в аккумуляторе находится код печатаемого символа, а в регистровой паре BC - координаты левого верхнего пиксела его шаблона. Следующая часть программы выполняет печать, а точнее "копирование" символа по точкам.

```
AB2D  5F       LD   E,A       ;Для рассчета адреса шаблона сим-
AB2E  1600     LD   D,#00     ;вола в символьном наборе, надо
                              ;рассчитать дистанцию от начала
                              ;символьного набора. Мы установили
                              ;в DE порядковый номер символа.
AB30  2160AB   LD   HL,#AB60 ;В HL - значение CHARS для упако-
                              ;ванного символьного набора.
AB33  19       ADD  HL,DE     ;Добавив к нему пятикратное зна-
AB34  19       ADD  HL,DE     ;чение порядкового номера,
AB35  19       ADD  HL,DE     ;получим адрес первого байта,
AB36  19       ADD  HL,DE     ;определяющего интересующий нас
AB37  19       ADD  HL,DE     ;шаблон.
AB38  1E05     LD   E,#05     ;Организация цикла из 5 шагов для
                              ;сканирования и побитного копиро-
                              ;вания 5 байтов шаблона.
AB3A  C5       PUSH BC        ;Сохранение координат для после-
                              ;дующего использования.
AB3B  7E       LD   A,(HL)    ;В аккумуляторе байт шаблона.
AB3C  B7       OR   A         ;Инициализация флагов.
AB3D  2811     JR   Z,#AB50   ;Если A=0, то не надо продолжать
                              ;печать этой линии и выполняется
                              ;переход на следующую линию.
```

```
AB3F  17      RLA              ;Если в А не ноль, то ротация для
                               ;определения необходимости печати
                               ;точки.
AB40  300B    JR   NC,#AB4D    ;Если бит равен нулю, то обход \
                               ;печати.
AB42  F5      PUSH AF          ;Сохранение на стеке
AB43  E5      PUSH HL          ;значений всех регистров,
AB44  D5      PUSH DE          ;так как они могут быть изменены
AB45  C5      PUSH BC          ;в процессе работы процедуры PLOT.
AB46  CDE522  CALL #22E5       ;Вызов процедуры PLOT.
AB49  C1      POP  BC          ;Восстановление со стека
AB4A  D1      POP  DE          ;значении всех регистров.
AB4B  E1      POP  HL          ;
AB4C  F1      POP  AF          ;
AB4D  0C      INC  C           ;Переход к следующему пикселу в
                               ;линии.
AB4E  18EC    JR   #AB3C       ;Повторение действий по печати
                               ;точки.
```

Сюда программа переходит, когда в аккумуляторе не остается ни одного включенного бита. Нет необходимости продолжать ротацию содержимого аккумулятора, если он пуст.

```
AB50  1D      DEC  E           ;Уменьшение счетчика линий.
AB51  C1      POP  BC          ;Прежнее значение координат печати
AB52  2804    JR   Z,#AB58     ;Если счетчик линий обнулен, то
                               ;переход на завершение процедуры.
AB54  05      DEC  B           ;Иначе - переход к печати следую-
                               ;щей пиксельной линии (координата
                               ;Y уменьшается на единицу).
AB55  23      INC  HL          ;Переход к следующему байту
                               ;в символьном наборе.
AB56  18E2    JR   #AB3A       ;Повторение действий по печати
                               ;пиксельнои линии.
```

Сюда программа переходит, когда печать символа по точкам завершена и необходимо провести финишную операцию - перейти к следующей позиции печати.

```
AB58  21FEAB   LD   HL.#ABFE ;В HL - адрес ячейки, в которой
                             ;задана колонка печати.
AB5B  34       INC  (HL)     ;Увеличение на единицу этого
                             ;значения - переход к следующему
                             ;знакоместу.
AB5C  C9       RET           ;Выход из процедуры печати.
```

Следующая часть программы - это утилита-упаковщик символьного набора. Исходный символьный набор должен быть загружен под адрес B000H (45056). Эта утилита выполняет "упаковку" символьного набора из стандартного формата 8*8 в формат 5*8 и переброску результата в адрес AC00H (44032).

```
AB80  2100B0   LD   HL.#B000 ;В HL - начало стандартного
                             ;символьного набора.
AB83  1100AC   LD   DE.#AC00 ;В DE - начало области для упако-
                             ;ванного символьного набора.
AB86  0660     LD   B.#60    ;Цикл для всех 96 символов.
AB88  C5       PUSH BC       ;Сохранение счетчика на стеке.
AB89  010500   LD   BC.#0005 ;Число перебрасываемых байтов.
AB8C  EDB0     LDIR          ;Переброска 5 байтов.
AB8E  23       INC  HL       ;Пропуск
AB8F  23       INC  HL       ;оставшихся
AB90  23       INC  HL       ;трех байтов.
AB91  C1       POP  BC       ;Восстановление со стека значения
                             ;счетчика для контроля.
AB92  10F4     DJNZ #AB88    ;Если эта процедура проделана не
                             ;со всеми 96 символами набора,
                             ;то повторение тех же действий.
AB94  C9       RET           ;Если все символы обработаны, то
                             ;выход.
```

Эта часть программы представляет собой утилиту-распаковщик. Она выполняет противоположную задачу - восстанавливает стандартный формат символьного набора.

```
ABB8  2100AC   LD   HL.#AC00 ;В HL - начало упакованного
                             ;символьного набора.
```

```
ABBB  1100B0   LD    DE,#B000 ;В DE - начало области для
                              ;стандартного символьного набора.
ABBE  0660     LD    B,#60    ;Цикл для всех 96 символов набора.
ABC0  C5       PUSH  BC       ;Сохранение счетчика на стеке.
ABC1  010500   LD    BC,#0005 ;Число перебрасываемых байтов.
ABC4  EDB0     LDIR           ;Переброска 5 байтов шаблона.
ABC6  AF       XOR   A        ;Обнуление
ABC7  12       LD    (DE),A   ;при помощи
ABC8  13       INC   DE       ;аккумулятора
ABC9  12       LD    (DE),A   ;трех
ABCA  13       INC   DE       ;оставшихся байтов
ABCB  12       LD    (DE),A   ;символьного
ABCC  13       INC   DE       ;набора.
ABCD  C1       POP   BC       ;Восстановление со стека значения
                              ;счетчика для контроля.
ABCE  10F0     DJNZ  #ABC0    ;Если эта процедура проделана не
                              ;со всеми символами набора, то
                              ;повторение тех же действий.
ABD0  C9       RET            ;Если все символы обработаны,
                              ;то выход.
```

Программные переменные, используемые в работе.

```
ABFE  00       DEFB           ;Колонка печати символа.
ABFF  00       DEFB           ;Строка печати символа.
```

Приведенная процедура печати может вызываться из БЕЙСИКа или непосредственно из машинного кода, когда происходит выполнение процедуры печати RST 10H. При этом, как уже говорилось, код печатаемого символа должен находиться в аккумуляторе. Но перед этим необходимо привязать новую процедуру печати к какому-нибудь потоку. Для этого надо предусмотреть инициализирующую подпрограмму, которая будет это выполнять.

Бейсик-программа, приведенная в листинге 2.8.2. демонстрирует работу новой процедуры печати.

Листинг 2.8.2.

```
  1 GO TO 10
  2 BORDER 7: PAPER 7: INK 0: CLEAR 40000
  3 LOAD "#4"CODE 43776
  4 LOAD "5*5.fnt"CODE 45056
 10 RANDOMIZE 43776: POKE 23588,PEEK 23670:
    POKE 23589,PEEK 23671
 20 POKE 23590,196: POKE 23591,21:
    POKE 23592,83
 30 RANDOMIZE (23588-(PEEK 23631+256*PEEK 23632)+1+65536)
 40 POKE 23582,PEEK 23670: POKE 23583,PEEK 23671
 90 RANDOMIZE USR 43904
100 LET X=0: LET Y=0: GO SUB 1000
110 FOR A=32 TO 127: PRINT #4;CHR$ A: NEXT A
120 GO TO 110
1000 POKE 44030,X: POKE 44031,Y: RETURN
```

Автостарт программы происходит со строки 2, после чего загружаются блоки кодов: "#4"CODE - это сама процедура печати, представленная выше, и "5*5.fnt"CODE - это исходный символьный набор стандартного формата 8*8, заготовленный при помощи графического редактора или редактора символов. Формирование нового канала, подключенного к потоку #4, выполняется в строках 10...40. Адрес процедуры плотной печати (43776=AB00H) задается в строке 10.

Строки 10 и 20 задают информацию по каналу:
- строка 10 - процедуру #PRINT;
- строка 20 - процедуру #INPUT.

Адреса обоих этих процедур должны быть заданы для любого канала. Так как для вывода на экран предусмотрена только процедура PRINT#, то для INPUT в строке 20 задается адрес 15C4H, как для основного экрана. Это обеспечивает вывод сообщения об ошибке: "Invalid stream" при попытке сделать INPUT #4. Строка 30 обеспечивает правильную работу инициализирующей подпрограммы, независимо от наличия или отсутствия у Вас интерфейса дис-

ковода. Необходимое смещение информации о каналах рассчитывается, учитывая значение системной переменной CHANS.

Строка 90 "упаковывает" символьный набор стандартного формата 8*8 в формат 5*8 и переносит его в рабочую область, под адрес ACOOH (44032).

Здесь надо сказать о том, что эта БЕЙСИК-программа всего лишь демонстрирует возможности печати и потому все подгоняется под простоту внесения изменении. Если Вы захотите внести изменения в символьный набор, то легко сможете это сделать при помощи графического редактора, так как исходный ("неупакованный") символьный набор всегда у Вас под рукой - файл "5*5fnt"CODE. Вы можете его легко изменить и проанализировать результаты. Когда Вы решите, что в таком виде символьный набор Вас устраивает, то можно сохранить "упакованный" символьный набор на ленте вместе с самой процедурой печати (так как сам по себе он все равно смысла не имеет). Это можно сделать командой: SAVE "#4"CODE 43776,736. К блоку кодов длиной 256 байтов добавляются по 5 байтов для всех 96 символов.

```
B Integer out of range. 110:2
```

Рис 16

Строка 100 при помощи подпрограммы 1000 задает начальную позицию печати: 0 строка и 0 колонка, т. е. левый верхний угол экрана. Строка 110 выполняет демонстрационную печать по потоку #4, который подключен к новой процедуре печати. По достижении последнего знакоместа в последней строке печать прекращается с сообщением об ошибке, как это предусмотрено, если позиция печати выходит за установленные пределы. Результат работы программы показан на Рис. 16.

Исходный символьный набор, как уже говорилось, Вы можете сформировать при помощи редактора символьного набора или графического редактора. Но если Вы по каким-то причинам не можете поступить таким образом, то ниже приводится дамп уже упакованного символьного набора (480 байт).

```
AC00: 00 00 00 00 00 20 20 20 :0C
AC08: 00 20 50 50 00 00 00 50 :C4
AC10: F8 50 F8 50 70 28 70 A0 :F4
AC18: 70 C8 D0 20 58 98 60 60 :9C
AC20: A8 90 68 20 40 00 00 00 :CC
AC28: 10 20 20 20 10 40 20 20 :D4
AC30: 20 40 50 20 70 20 50 00 :8C
AC38: 20 70 20 00 00 00 00 20 :B4
AC40: 40 00 00 78 00 00 00 00 :A4
AC48: 00 60 60 08 10 20 40 80 :AC

AC50: 60 90 90 90 60 20 60 20 :0C
AC58: 20 70 60 90 20 40 F0 E0 :B4
AC60: 10 60 10 E0 90 90 F0 10 :8C
AC68: 10 F0 80 E0 10 E0 60 80 :44
AC70: E0 90 60 F0 10 20 40 40 :8C
AC78: 60 90 60 90 60 60 90 70 :C4
AC80: 10 60 00 20 00 20 00 00 :DC
AC88: 20 00 20 40 00 20 40 20 :34
AC90: 00 00 70 00 70 00 00 40 :5C
AC98: 20 40 00 60 10 20 00 20 :54
```

```
ACA0: 70 B0 B0 80 70 60 90 90 :8C
ACA8: F0 90 E0 90 E0 90 E0 60 :F4
ACB0: 90 80 90 60 E0 90 90 90 :EC
ACB8: E0 F0 80 E0 80 F0 F0 80 :74
ACC0: E0 80 80 60 80 B0 90 60 :CC
ACC8: 90 90 F0 90 90 70 20 20 :54
ACD0: 20 70 30 10 10 90 60 90 :DC
ACD8: A0 C0 A0 90 80 80 80 80 :14
ACE0: F0 88 D8 A8 88 88 90 D0 :F4
ACE8: B0 90 90 60 90 90 90 60 :D4

ACF0: E0 90 E0 80 80 60 90 90 :6C
ACF8: B0 70 E0 90 E0 A0 90 60 :A4
AD00: 80 60 10 60 70 20 20 20 :CD
AD08: 20 90 90 90 90 60 90 90 :95
AD10: 90 F0 60 A8 A8 A8 F8 50 :DD
AD18: 90 90 60 90 90 88 50 20 :5D
AD20: 20 20 F0 10 60 80 F0 30 :0D
AD28: 20 20 20 30 80 40 20 10 :55
AD30: 08 60 20 20 20 60 20 70 :95
AD38: 20 20 20 00 00 00 00 F8 :3D

AD40: 30 40 F0 40 F0 60 90 90 :FD
AD48: F0 90 E0 80 E0 90 E0 A0 :C5
AD50: A0 A0 F0 10 30 50 50 50 :5D
AD58: F8 FC 80 E0 80 F0 70 A8 :D5
AD60: A8 70 20 70 40 40 40 40 :B5
AD68: 90 90 60 90 90 90 90 B0 :85
AD70: D0 90 60 90 B0 D0 90 90 :0D
AD78: A0 C0 A0 90 30 50 50 50 :D5
AD80: 90 88 D8 A8 88 88 90 90 :F5
AD88: F0 90 90 60 90 90 90 60 :B5

AD90: F0 90 90 90 90 70 90 70 :DD
AD98: 50 90 E0 90 E0 80 80 60 :D5
ADA0: 90 80 90 60 70 20 20 20 :1D
ADA8: 20 90 90 70 10 60 A8 A8 :C5
ADB0: 70 A8 A8 E0 90 E0 90 E0 :DD
```

```
ADB8: 80 80 E0 90 E0 90 90 D0 :A5
ADC0: B0 D0 60 90 20 90 60 88 :75
ADC8: A8 A8 A8 F8 E0 10 70 10 :D5
ADD0: E0 A8 A8 A8 F8 08 90 90 :75
ADD8: 70 10 10 C0 40 60 50 60 :25
```

Число в конце каждой строки после двоеточия - это контрольная сумма строки. Она поможет Вам при наборе этого блока кодов. Для набора блока кодов можно воспользоваться программой шестнадцатиричного ввода, которая приведена в листинге 2.8.3. Данные вводятся непосредственно при помощи INPUT. Периодически, вводя "S", Вы имеете возможность сохранить на ленте то, что уже введено. Приведенные выше коды напечатаны вразрядку, да еще с разделителями ":". Это сделано для удобочитаемости данных. Вам при наборе надо будет вводить их подряд, например для первой строки:

AC000000000000202020 0C [ENTER]

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

Листинг 2.8.3

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

```
1060 LET a(1)=FN A(b$)
1070 NEXT 1
1080 LET add=a(1)*256+a(2)
1090 LET sum=a(1)+a(2)
1100 FOR 1=3 TO 10
1110 LET b$=h$(2*1-1 TO 2*1)
1120 LET a(1)=FN A(b$)
1130 LET sum=sum+a(1)
1140 POKE add, a(1)
1150 LET add=add+1
1160 NEXT 1
1170 LET b$=h$(21 TO )
1180 LET cs=FN A(b$)
1190 LET cs1=sum-256*INT (sum/256)
1200 IF cs<>cs1 THEN  PRINT a$; INVERSE 1;" ERROR ! ":
 BEEP 1,0: LET add=add-8: GO TO 1000
1210 PRINT a$( TO 4): GO TO 1000
2000 SAVE "5*5. fnt"CODE 44032,480
2010 VERIFY "5*5. fnt"CODE
2020 CLS : PRINT a$( TO 4): GO TO 1000
```

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

2. 8. 2. Шрифты размером более, чем 8X8.

Аналогично тому, как была организована процедура плотной печати, когда размер символа был 5*5 пикселов, может быть организована процедура печати, когда размер одного символа превышает 8*8 пикселов. Это может потребоваться для печати символов повышенного качества (например, "объемный" шрифт с тенью).

Самое простое - использовать для печати одного символа 4

знакоместа: два по горизонтали и два по вертикали. Однако, для 96 символов набора потребуется 768*4=3072 байта памяти! Да и сами символы размером 16*16 пикселов могут оказаться слишком крупными для размещения достаточного количества информации на одном экране. Вот, если бы можно было печатать символами размером крупнее, чем 8*8, но мельче, чем 16*16, т.е. произвольными символами.

В принципе, раз мы однажды уже реализовали процедуру плотной печати, то, используя те же принципы, можно реализовать и другую процедуру, назовем ее процедурой качественной печати.

Процедура качественной печати выполняет печать символами с шагом в высоту 13 пикселов и в ширину 12 пикселов. При этом на экране умещается 176/13=13 строк и в каждой строке может быть напечатано по 256/12=21 символу. Условимся, аналогично тому, как это было сделано в процедуре плотной печати, что высота символа не может превышать 11 пикселов, а 2 пустые линии будем использовать для разделения строк. Шаблон каждого символа в символьном наборе будет состоять из двух половин: левой и правой. Каждая половина будет состоять из 11 байтов, то есть шаблон одного символа будет занимать 22 байта памяти. Всего на такой символьный набор будет отведено 22*96=2112 байтов. Это заметно меньше, чем в том случае, когда используется печать 16*16 пикселов. По причинам, указанным ранее, мы не будем заниматься еще и экономией недоиспользуемых 4 младших битов в правой половине символа.

Условимся, что расположение байтов в символьном наборе, определяющих шаблон символа, будет таким, как показано на Рис. 17. Можно применить и другой вариант, - это непринципиально, однако надо что-то принять за основу и в дальнейшем придерживаться этого условия. Такой символьный набор может быть сформирован при помощи обычных символьных наборов, если один символ исполнять из четырех стандартных символов, причем расположены они должны быть определенным образом, как показано на Рис. 18.

При помощи таких четверок задается каждый символ для ка-

чественной печати. Таким образом, для того, чтобы сформировать символьный набор из 96 символов, нам нужны четыре стандартных символьных набора. В первом - будут заданы 24 символа с кодами с 32 ("пробел") по 55 ("7"); во втором - с 56 ("8") по 79 ("O"); в третьем - с 80 ("P") по 103 ("g"); в четвертом - с 104 ("h") по 127 ("копирайт").

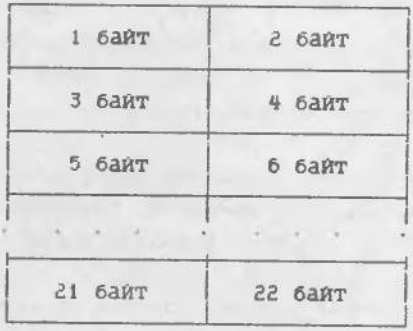


Рис. 17

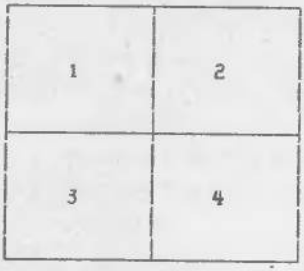


Рис. 18.

К вопросу о формировании такого символьного набора мы еще вернемся, а пока предположим, что он уже сформирован и расположен, как и в процедуре плотной печати, начиная с адреса AC00H (44032). Значение CHARS для такого символьного набора будет равно: 44032-22*32=43328 (A940H).

Процедура качественной печати приведена в листинге 2.8.4. Она в основном похожа на процедуру плотной печати, имеет аналогичное построение, расположена в тех же адресах и имеет те же входные параметры.

Листинг 2.8.4

```
AB00  FE20      CP   #20      ;Проверка, не является ли печатае-
                               ;мый код управляющим символом.
AB02  D8        RET  C         ;Если да, то возврат из процедуры
                               ;печати.
AB03  FE80      CP   #80      ;Проверка, не является ли печатае-
                               ;мый код токеном ключевого слова
                               ;или символом блочной графики.
AB05  D0        RET  NC        ;Если да, то возврат из процедуры.
```

Следующая часть программы вычисляет координаты в пикселах места для печати символа по заданным координатам в знакоместах. Последние заданы при помощи POKE в ячейках ABFEH (координата X) и ABFFH (координата Y). Результат рассчета будет получен в регистре BC.

```
AB06  F5        PUSH AF        ;Сохранение на стеке кода печатае-
                               ;мого символа.
AB07  21FFAB    LD   HL,#ABFF  ;В HL засылается адрес ячейки, в
                               ;которой находится номер строки
                               ;печати.
AB0A  7E        LD   A,(HL)    ;Значение, содержащееся там, срав-
AB0B  FE0D      CP   #0D       ;нивается с максимально допустимым
                               ;(0DH=13 строк).
AB0D  D29F1E    JP   NC,#1E9F  ;Если это значение превышено, то
                               ;переход на процедуру REPORT B в
                               ;ПЗУ для вывода сообщения об
                               ;ошибке "Integer out of range".
AB10  2B        DEC  HL        ;Теперь в HL адрес ячейки, в кото-
                               ;рой находится значение колонки
                               ;печати.
AB11  7E        LD   A,(HL)    ;Значение, содержащееся там, срав-
```

```
AB12  FE15     CP   #15       ;нивается с максимально допустимым
                               ;(15H=21 символ в строке).
AB14  3806     JR   C,#AB1C   ;Если результат не превышен, то
                               ;переход на продолжение процедуры
                               ;печати.
AB16  AF       XOR  A         ;Иначе - переход на новую строку.
AB17  77       LD   (HL),A    ;Для этого обнуляется счетчик
AB18  23       INC  HL        ;колонок, а счетчик строк
AB19  34       INC  (HL)      ;увеличивается на единицу.
AB1A  18EE     JR   #AB0A     ;После этого возврат на контроль
                               ;счетчика строк, не превышен ли
                               ;предел.
AB1C  4F       LD   C,A       ;Теперь значение колонки печати,
                               ;находится и в регистре C.
AB1D  87       ADD  A,A       ;Таким способом производится
AB1E  81       ADD  A,C       ;умножение этого значения на 12
AB1F  87       ADD  A,A       ;для рассчета координаты
AB20  87       ADD  A,A       ;в пикселах.
AB21  4F       LD   C,A       ;Полученная координата X сохраня-
                               ;ется в регистре C.
AB22  23       INC  HL        ;Переходим к адресу, где задана
                               ;колонка печати.
AB23  7E       LD   A,(HL)    ;Взяли ее значение в аккумулятор.
AB24  47       LD   B,A       ;Таким способом
AB25  87       ADD  A,A       ;производится умножение
AB26  80       ADD  A,B       ;этого значения на 13
AB27  87       ADD  A,A       ;для рассчета
AB28  87       ADD  A,A       ;координаты в пикселах
AB29  80       ADD  A,B       ;т. к. шаг строк - 13 пикселов.
AB2A  47       LD   B,A       ;Полученное значение Y сохраняется
                               ;в регистре B.
AB2B  3EAD     LD   A,#AD     ;Его надо вычесть из числа 173,
AB2D  90       SUB  B         ;так как отсчет координаты в
                               ;пикселах ведется из левого
                               ;нижнего угла экрана.
```

Аналогично предыдущей процедуре, выбрана величина 173, а не 175, т. к. тогда две пустые верхние пиксельные линии будут

отделять текст от бордюра.

```
AB2E  47       LD   B,A       ;Скорректированное значение Y
                               ;сохраняется в регистре B.
AB2F  F1       POP  AF        ;Восстановление кода печатаемого
                               ;символа.
AB30  5F       LD   E,A       ;Для расчета адреса шаблона сим-
AB31  1600     LD   D,#00     ;вола в символьном наборе, надо
                               ;рассчитать его смещение от начала
                               ;символьного набора. В DE устанав-
                               ;ливаем порядковый номер символа.
AB33  D5       PUSH DE        ;Через стек это значение
AB34  E1       POP  HL        ;переписывается в HL.
AB35  29       ADD  HL,HL     ;Таким способом
AB36  29       ADD  HL,HL     ;порядковый номер символа
AB37  19       ADD  HL,DE     ;умножается на 22, так как
AB38  29       ADD  HL,HL     ;это число является шагом,
AB39  19       ADD  HL,DE     ;определяющим число байтов на 1
AB3A  29       ADD  HL,HL     ;символ в символьном наборе.
AB3B  1140A9   LD   DE,#A940  ;В DE - значение CHARS для упако-
                               ;ванного символьного набора.
AB3E  19       ADD  HL,DE     ;Сложив с ним величину, получен-
                               ;ную в HL, получим адрес первого
                               ;из 22 байтов, определяющих шаблон
                               ;символа.
AB3F  160B     LD   D,#0B     ;Организация цикла из 11 повторений
                               ;для 11 пиксельных линий.
AB41  1E02     LD   E,#02     ;Организация цикла из 2 повторений
                               ;для левой и правой половины символа.
AB43  C5       PUSH BC        ;Двукратное сохранение координат на
AB44  C5       PUSH BC        ;стеке для простоты последующих
                               ;рассчетов.
AB45  7E       LD   A,(HL)    ;В аккумуляторе байт шаблона .
AB46  B7       OR   A         ;Инициализация флагов (в частности
                               ;для нас важен принудительный
                               ;сброс флага переноса.
AB47  2811     JR   Z,#AB5A   ;Если в A ноль, то нет необходи-
                               ;мости продолжать работу по печати
```

```
                                    ;этой части символа и выполняется
                                    ;переход к следующему байту.
AB49  17        RLA                 ;Если в А не ноль, то ротацией
                                    ;через флаг переноса определяем
                                    ;необходимость печати данной
                                    ;точки.
AB4A  300B      JR    NC,#AB57      ;Если флаг сброшен, то обход
                                    ;печати.
AB4C  F5        PUSH  AF            ;Сохранение на стеке значений
AB4D  E5        PUSH  HL            ;регистров, т.к. они могут быть
AB4E  D5        PUSH  DE            ;изменены в процессе
AB4F  C5        PUSH  BC            ;работы процедуры PLOT.
AB50  CDE522    CALL  #22E5         ;Вызов процедуры PLOT.
AB53  C1        POP   BC            ;Восстановление со стека
AB54  D1        POP   DE            ;значений регистров.
AB55  E1        POP   HL            ;
AB56  F1        POP   AF            ;
AB57  0C        INC   C             ;Переход к следующему пикселу.
AB58  18EC      JR    #AB46         ;Повторение действий по печати
                                    ;точек.
```

Сюда программа проходит, когда в аккумуляторе не остается ни одного включенного бита. Нет необходимости продолжать сдвигать содержимое аккумулятора, если он пуст.

```
AB5A  C1        POP   BC            ;Восстановление координат со
                                    ;стека.
AB5B  1D        DEC   E             ;Переход к правой половине
                                    ;шаблона символа.
AB5C  2807      JR    Z,#AB65       ;Если правая половина уже напе-
                                    ;чатана, то переход к следующей
                                    ;пиксельной линии символа.
AB5E  79        LD    A,C           ;Иначе - увеличение
AB5F  C608      ADD   A,#08         ;координаты X на 8 пикселов.
AB61  4F        LD    C,A           ;Новое значение - опять в С.
AB62  23        INC   HL            ;Переход к следующему байту
                                    ;символьного набора.
AB63  18DF      JR    #AB44         ;Повторение действий для правой
```

```
                                    ;половины символа.
AB65  C1        POP   BC            ;Восстановление координат со
                                    ;стека.
AB66  15        DEC   D             ;Проверка счетчика пиксельных
                                    ;линий.
AB67  2804      JR    Z.#AB6D       ;Если достигнут ноль, то переход
                                    ;на завершение процедуры.
AB69  05        DEC   B             ;Переход к следующей пиксельной
                                    ;линии (координата Y уменьшается.)
AB6A  23        INC   HL            ;Переход к следующему байту
                                    ;шаблона.
AB6B  18D4      JR    #AB41         ;Повторение действий по печати
                                    ;пиксельной линии.
```

Сюда программа проходит, когда печать символа по точкам завершена и необходимо провести финишную операцию - перейти к следующей позиции печати.

```
AB6D  21FEAB    LD    HL.#ABFE      ;В HL - адрес ячейки, в которой
                                    ;задана колонка печати.
AB70  34        INC   (HL)          ;Увеличение на единицу этого зна-
                                    ;чения - переход к следующему
                                    ;знакоместу.
AB71  C9        RET                 ;Выход из процедуры печати.
```

Следующая часть программы - это утилита-упаковщик символьного набора. Исходные четыре символьных набора должны быть загружены под адреса: B500H (46336), B800H (47104), BB00H (47872) и BE00H (48640). Эта утилита выполняет "упаковку" символьного набора и переброску результата в адрес AC00H (44032).

```
AB80  DD2100B5 LD    IX.#B500      ;В IX - начало первого стандарт-
                                    ;ного символьного набора.
AB84  2100AC    LD    HL.#AC00      ;В HL - начало области для упако-
                                    ;ванного символьного набора.
AB87  0660      LD    B.#60         ;Цикл для всех 96 символов набора.
AB89  C5        PUSH  BC            ;Сохранение счетчика на стеке.
AB8A  0608      LD    B.#08         ;Цикл для восьми верхних линий
```

```
                                    ;символа.
AB8C  DD7E00    LD    A,(IX+0)      ;В аккумуляторе - байт левой
                                    ;половины.
AB8F  77        LD    (HL),A        ;Упаковка его в новый симв. набор.
AB90  23        INC   HL            ;Переход к следующему адресу.
AB91  DD7E08    LD    A,(IX+8)      ;В аккумуляторе - байт правой
                                    ;половины.
AB94  77        LD    (HL),A        ;Упаковка его в новый симв. набор.
AB95  23        INC   HL            ;Переход к следующему адресу.
AB96  DD23      INC   IX            ;Переход к следующему адресу
                                    ;в исходном символьном наборе.
AB98  10F2      DJNZ  #AB8C         ;Повторение для всех 8 пиксельных
                                    ;линий, определяющих верхнюю часть
                                    ;символа.
AB9A  110800    LD    DE,#0008      ;При помощи смещения в 8 байт вы-
AB9D  DD19      ADD   IX,DE         ;полняется переход к адресам, оп-
                                    ;ределяющим нижнюю часть символа.
AB9F  0603      LD    B,#03         ;Здесь организуется цикл из трех
                                    ;шагов, т.к. 8 уже сделано и до 11
                                    ;осталось сделать три.
ABA1  DD7E00    LD    A,(IX+0)      ;Эта часть программы
ABA4  77        LD    (HL),A        ;полностью идентична
ABA5  23        INC   HL            ;той, которая расположена
ABA6  DD7E08    LD    A,(IX+8)      ;в адресах AB8CH ... AB98H.
ABA9  77        LD    (HL),A
ABAA  23        INC   HL
ABAB  DD23      INC   IX
ABAD  10F2      DJNZ  #ABA1
ABAF  110D00    LD    DE,#000D      ;При помощи смещения в 13 байтов
ABB2  DD19      ADD   IX,DE         ;выполняетсяется переход к следу-
                                    ;ющей группе байтов, определяющих
                                    ;четверку символов в исходном
                                    ;символьном наборе.
ABB4  C1        POP   BC            ;Восстановление счетчика символов
ABB5  10D2      DJNZ  #AB89         ;со стека и его проверка. Если не
                                    ;достигнут конец, то повтор тех же
                                    ;деиствий для следующего символа.
ABB7  C9        RET                 ;Иначе - выход из процедуры.
```

Следующая часть программы - это утилита-распаковщик символьного набора. Упакованный символьный набор должен быть расположен начиная с адреса AC00H (44032). Четыре стандартных символьных набора будут получены в адресах: B500H (46336), B800H (47104), BB00H (47872) и BE00H (48640).

Стартовая часть программы обнуляет область для получаемых символьных наборов. Это необходимо сделать, так как при работе распаковщик "перескакивает" через неиспользуемые ячейки, в которых могла остаться какая-то информация.

```
ABB8  2100B5    LD    HL,#B500 ;В HL - начало области для перво-
                               ;го стандартного символьного
                               ;набора.
ABBB  3600      LD    (HL),#00 ;Обнуление первого байта.
ABBD  1101B5    LD    DE,#B501 ;Подготовка параметров
ABC0  01FF0B    LD    BC,#0BFF ;для LDIR.
ABC3  EDB0      LDIR           ;Обнуление памяти при помощи LDIR.
```

Далее - собственно процедура, противоположная упаковщику.

```
ABC5  DD2100B5 LD    IX,#B500 ;В IX - начало области для перво-
                               ;го стандартного символьного
                               ;набора.
ABC9  2100AC    LD    HL,#AC00 ;В HL - начало упакованного
                               ;символьного набора.
ABCC  0660      LD    B,#60    ;Цикл для всех 96 символов набора.
ABCE  C5        PUSH  BC       ;Сохранение счетчика на стеке.
ABCF  0608      LD    B,#08    ;Цикл для восьми верхних линий
                               ;шаблона.
ABD1  7E        LD    A,(HL)   ;В аккумуляторе байт упакованного
                               ;символьного набора.
ABD2  DD7700    LD    (IX+0),A ;Перенесение его в стандартный
                               ;символьный набор.
ABD5  23        INC   HL       ;Переход к следующему адресу.
ABD6  7E        LD    A,(HL)   ;Следующий байт упакованного
                               ;символьного набора.
ABD7  DD7708    LD    (IX+8),A ;Перенесение его в стандартный
```

```
                                    ;символьный набор.
ABDA  23        INC   HL            ;Переход к следующим адресам
ABDB  DD23      INC   IX            ;символьных наборов.
ABDD  10F2      DJNZ  #ABD1         ;Повторение для всех 8 верхних
                                    ;линий.
ABDF  110800    LD    DE,#0008      ;При помощи смещения в DE, увели-
ABE2  DD19      ADD   IX,DE         ;чение на 8 байтов адреса в
                                    ;стандартном символьном наборе.
ABE4  0603      LD    B,#03         ;Цикл для трех нижних линий
                                    ;шаблона.
ABE6  7E        LD    A,(HL)        ;Эта часть программы
ABE7  DD7700    LD    (IX+0),A      ;полностью идентична
ABEA  23        INC   HL            ;программе в адресах
ABEB  7E        LD    A,(HL)        ;ABD1H ... ABDDH
ABEC  DD7708    LD    (IX+8),A
ABEF  23        INC   HL
ABF0  DD23      INC   IX
ABF2  10F2      DJNZ  #ABE6
ABF4  110D00    LD    DE,#000D      ;При помощи смещения в DE, увели-
ABF7  DD19      ADD   IX,DE         ;чение адреса на 13 байтов, что
                                    ;обеспечивает переход к следующей
                                    ;четверке символов.
ABF9  C1        POP   BC            ;Восстановление со стека значения
ABFA  10D2      DJNZ  #ABCE         ;счетчика символов и его проверка.
                                    ;Если не достигнут конец, то пов-
                                    ;торение действий для следующего
                                    ;символа.
ABFC  C9        RET                 ;Если конец достигнут, то выход.
```

Системные ячейки, используемые в работе программы.

```
ABFE  00        DEFB                ;Колонка печати символа.
ABFF  00        DEFB                ;Строка печати символа.
```

Если мы хотим, чтобы эта новая процедура качественной печати могла работать из БЕЙСИКа по команде PRINT# или из машинного кода по команде RST 10H, то нам необходимо открыть новый

канал. Пусть, например, он будет подключен к потоку #5.

Бейсик-программа, приведенная в листинге 2.8.5, демонстрирует работу этои процедуры печати. Это только демонстрационная программа, поэтому в строке 4 в неи предусмотрена загрузка только одного стандартного символьного набора - второго, определяющего символы с "8" по "O".

Листинг 2.8.5.

```
  1 GO TO 10
  2 BORDER 7: PAPER 7: INK 0: CLEAR 40000
  3 LOAD "#5"CODE 43776
  4 LOAD "11*12.fnt"CODE 47104,768
 10 RANDOMIZE 43776: POKE 23593,PEEK 23670:
    POKE 23594,PEEK 23671
 20 POKE 23590,196: POKE 23591,21: POKE 23592,83:
    POKE 23595,PEEK 23853: POKE 23596,PEEK 23584:
    POKE 23597,PEEK 23855
 30 RANDOMIZE (23593-(PEEK 23631+256*PEEK 23632)+1+65536)
 40 POKE 23584,PEEK 23670: POKE 23585,PEEK 23671
 90 RANDOMIZE USR 43904
100 LET x=0: LET y=0: GO SUB 1000
110 PRINT #5;"89:;<=>?@ABCDEFGHIJK"
120 GO TO 110
1000 POKE 44030,x: POKE 44031,y: RETURN
```

```
B Integer out of range, 110:1
```

Рис. 19

Теперь несколько слов о том, как сформировать символьный набор для процедуры качественной печати. Это можно сделать опять же при помощи графического редактора "ARTSTUDIO", только теперь объем работы будет больше. Каждый символ "качественного" символьного набора сначала надо сформировать, пользуясь лупой с увеличением x8 (режим MAGNIFY X 8). Начинать работу надо в левом верхнем углу экрана, так, как это показано на Рис. 20. Надо строго следить за тем, чтобы символы располагались точно с шагом 16*16 пикселов. Всего их должно быть 24, согласно распределению по четырем символьным наборам, которое было приведено выше.

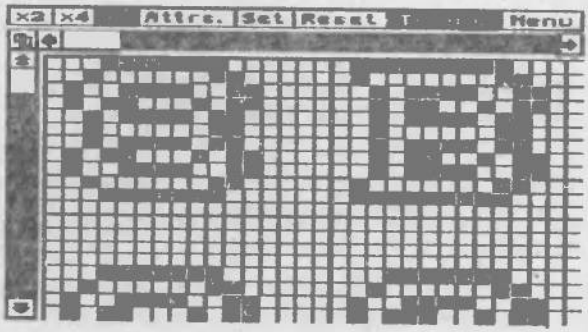


Рис. 20

Когда работа будет закончена, картина должна иметь примерно такой вид, как показано на Рис. 21.

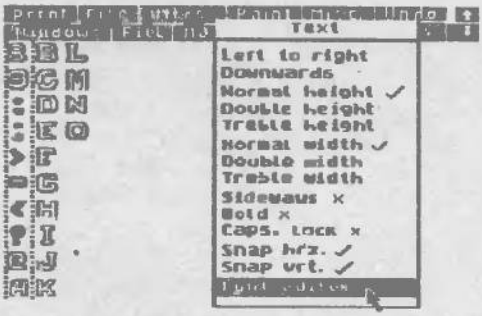


Рис. 21

Здесь дан второй символьный набор из четырех. Теперь, задав окно так, как это показано на Рис. 21 , войдем в меню "Font editor" и выполним операцию "Capture font", как это показано на Рис. 22.

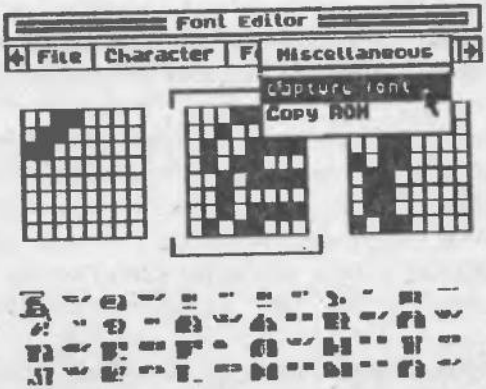


Рис. 22

Указатель текущего символа должен быть установлен на первый символ. Необходимое условие: окно должно быть задано точно шириной в 16 пикселов, иначе фрагменты символов будут считаны со смещением. Такую же процедуру надо повторить для второго и третьего столбцов символов (см. Рис. 21), каждый раз передвигая указатель текущего символа на новое место (см. Рис. 22). После окончания работы символьный набор примет вид, показанный на Рис. 22. Теперь он готов к упаковке.

3. ВЕКТОРНАЯ ГРАФИКА

Мы рассмотрели основы растровой графики и убедились, что основным элементом растровых изображений является обыкновенная точка (пиксел экрана). Все изображения получаются путем копирования пикселов заранее заданного шаблона в экранный файл.

Принципы векторной графики существенно отличаются от принципов графики растровой. Здесь основным элементом изображения является прямая линия, а правильнее говоря отрезок прямой. Все графические объекты на экране получаются в виде ажурных конструкций, созданных из набора отрезков. С их помощью могут получаться как плоские фигуры, так и проекции трехмерных тел.

Вы, уважаемые читатели, обратили, конечно, внимание на то, что векторная графика в играх выглядит одноцветно, угловато и художественными достоинствами очевидно не отличается. Так почему же такие игры пользуются огромным успехом, с чем он связан?

Да, конечно, векторная графика выглядит на экране победнее, чем многоцветная растровая графика, но у нее есть два огромных преимущества. Во-первых, это очень быстрая графика. Цикл освежения экрана и перестроения изображения происходит намного быстрее, чем в программах с растровой графикой. Во-вторых, это вычисляемая графика, то есть не надо хранить в памяти компьютера заранее подготовленные экраны. Все изображения рассчитываются по заданным алгоритмам и практически никогда не повторяются. Благодаря этому Вы можете иметь в таких программах тысячи планетных систем, десятки возможных кораблей противника и нескончаемое разнообразие игровых ситуации.

Вспомним программу "ELITE". Да, конечно, нужно иметь воображение, чтобы принять угловатую "морковку" на экране Вашего монитора за роскошный корабль "Fer-de Lance", нашпигованный чудесами науки и техники и отделанный изнутри лучшими породами дерева и самыми дорогими материалами. Но зато когда он вращается вокруг всех собственных осей и при этом летит в пространст-

ве, изменяя свои координаты относительно Вашего корабля, а Вы вместе с ним при этом тоже перемещаетесь и маневрируете относительно планеты, звезды, станции и прочих кораблей, и этот клубок пронзают залпы лазеров, в нем летят и находят свою цель ракеты, здесь Вы забываете обо всем - и о "морковке" и о черно-белой графике. Перед Вами реальный, хорошо вооруженный противник - это вызов Вашему мастерству. Динамика игры, острота схватки и неповторяемость ситуации делают возможным для Вас эффект реального присутствия и Вам уже не нужно художественное впечатление от богатства красок. Ваш мозг, увлеченный переживаниями, сам домыслит столько, сколько ему надо.

Итак, векторная графика - вычисляемая. Таким образом, основной программный аппарат, которым Вы должны располагать - это процедуры для изображения отрезков прямых между точками с заданными координатами, математические процедуры, рассчитывающие экранные координаты концов отрезков, в основе которых лежат известные формулы из геометрии и тригонометрии, а также некоторые процедуры для организации специальных приемов, таких как клиппирование, масштабирование, сокрытие линий невидимого контура и затушевывание поверхностей, находящихся в тени (если задано направление источника освещения).

Вот на этих технических приемах мы и остановимся. Единственное, что мы не будем рассматривать - это технику затушевывания теневых поверхностей, поскольку для того, чтобы определить, какая поверхность трехмерного тела находится в тени относительно заданного источника, необходимо применять математический аппарат из области аналитической геометрии, а этот курс ВУЗовский и не входит в программу средней школы, под которую мы в принципе подстраиваем наши книги. Те же читатели, которые знакомы с этой наукой, смогут подумав и сами реализовать необходимые алгоритмы, особенно если учесть, что процедуру для заполнения замкнутого контура мы уже дали в прошлой книге - "Элементарная графика". Проблема сводится фактически не к тому, как заштриховать контур, а к тому, чтобы определить только какой контур надо закрашивать, а какой - нет. Надо также учитывать и то, что эта техника очень редко применяется в динамичных играх

с быстроменяющеися ситуациеи. Этот прием больше подходит для медленных игр стратегического или адвентюрного направления.

Мы не будем также рассматривать здесь вопросов анимации изображений, хотя именно здесь векторная графика и проявляет свои основные достоинства, оставив их для рассмотрения в третьем томе нашеи серии, который будет называться "Динамическая графика".

3.1. Клиппирование изображений.

Выше мы сказали о том, что в основу векторноы графики положено изображение тел и поверхностей в виде отрезков прямых. И первое, о чем мы должны позаботиться - это об изображении ЛЮБЫХ отрезков прямых, а не только тех, которые нам позволяют изображать ограниченные размеры экрана компьютера.

Рассмотрим конкретную игровую ситуацию, знакомую каждому, кто хоть когда-нибудь играл в программу "Elite" фирмы Firebird Software или в другую подобную игру для космических ассов или хотя бы слышал о них. Предположим, что перед Вами появился враждебеный "Таргон" и открыл огонь на поражение. Ваши координаты примем за нулевые (X1=0; Y1=0) и оии совпадают с центром экрана (или с положением прицела Вашего корабля). Координаты "Таргона" (X2,Y2) находятся в пределах экрана (Рис. 23 а).

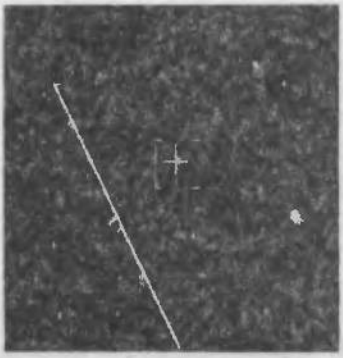


Рис. 23 а.

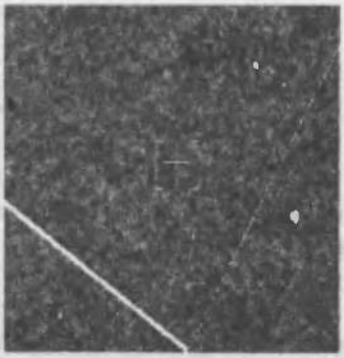


Рис. 23 б.

Что представляет с точки зрения программиста изображение вражеского лазерного луча? Это отрезок прямой линии, соединяющей точки X1,Y1 и X2,Y2. Нарисовать такой отрезок очень несложно. Можно воспользоваться процедурои ПЗУ, можно взять ту процедуру, которую мы рассмотрели в книге "Элементарная графика", можно, в конце концов, составить и свою, здесь нет никаких проблем.

Теперь попробуем развернуть свой корабль вправо. Враг покинул экран (Рис.236), его больше нет, мы спрятали "голову в песок" и что? Прекратился обстрел? Ничуть. Все также пронзают черноту экрана яркие вспышки лазеров невидимого "Таргона". Что представляет собой такая ситуация с точки зрения программиста? Только то, что программа изображает отрезки прямых между точками, одна из которых (Ваш корабль) находится в области экрана, а другая - нет. Это уже никакими процедурами ПЗУ и им подобными сделать нельзя. А ситуация эта, как Вы должно быть понимаете, очень распространена для программ, имеющих дело с векторной графикой и ее надо рассмотреть особо.

Для того, чтобы научиться рисовать что-либо в векторной графике, необходимо сначала научиться рисовать обычные прямые линии. Команда DRAW в БЕЙСИКе вполие может Вам нарисовать такую линию, если вся эта линия целиком укладывается на экране. А теперь представим себе, что будет, если она НЕ УКЛАДЫВАЕТСЯ на экране? Например, по командам:

```
PLOT 20,20
DRAW 250,250
```

Если Вы попробуете это сделать, то безусловно получите сообщение об ошибке "B Integer out of range" и программа прекратит работу.

Может быть, все-таки лучше было бы, если бы программа не прекращала работу, а изобразила хотя бы ту часть прямой, которая укладывается на экране. Эта концепция называется "клиппиро-

ванием" изображения. Т. е. из изображения "вырезается" та часть, которая может быть нарисована на экране без проблем.

Вполне возможен случай, когда вся Ваша предполагаемая линия не попадает на экран. В этом случае вся она будет клиппирована и ничего на экране вообще изображено не будет.

Давайте рассмотрим, как работает концепция клиппирования. Взгляните на рис. 24.

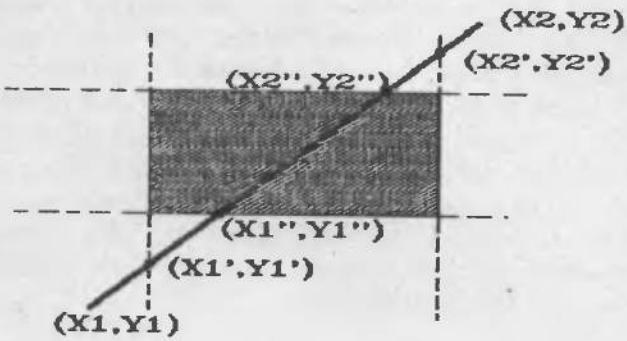


Рис. 24

Заштрихованный прямоугольник - это наш экран. Предположим, что Вы хотите изобразить отрезок прямой, соединяющей координаты X1,Y1 и X2,Y2. Чтобы клиппировать этот отрезок, мы должны так приблизить друг к другу координаты его концов, чтобы получить в конце концов отрезок, вписывающиися в пределы экрана. Эта операция выполняется в четыре приема.

Сначала мы отыскиваем точку X1'Y1', в которой наш отрезок пересекает левую границу экрана. Затем мы отыскиваем точку X1'',Y1'', в которой отрезок пересекает нижнюю границу экрана и потом, соответственно, точки X2',Y2' и X2'',Y2''. Как видите, отрезок от X1'',Y1'' до X2'',Y2'' полностью вписывается в пределы экрана компьютера.

Эта же технология должна работать и если клиппируемая линия будет располагаться по-другому, а не так, как показано на

Рис. 24. Т. е. где бы ни лежала точка X1, Y1, мы должны сначала рассчитать X1', Y1' и передвинуть точку к ближайшей правой или левой границе экрана ТОЛЬКО В ТОМ СЛУЧАЕ, если она лежит от экрана слева или справа. В противном случае ее следует оставить там, где она есть. После этого можно переходить к расчету X1'', Y1'', передвигая точку ТОЛЬКО ЕСЛИ ЭТО НЕОБХОДИМО (если она лежит ниже или выше пределов экрана). То же самое относится и к точке X2, Y2.

Необходимо предусмотреть варианты, когда вся линия лежит вне пределов экрана. Так, если координаты и начала и конца отрезка лежат слева от экрана, то клиппировать в этой линии нечего и изображать ее не надо. То же самое, если координаты обоих концов лежат справа, сверху или снизу от экрана.

Впрочем, могут быть и еще некоторые положения отрезка, при котором он не проходит через экран и которые идентифицировать гораздо более сложно. На Рис. 25 показаны примеры таких отрезков.

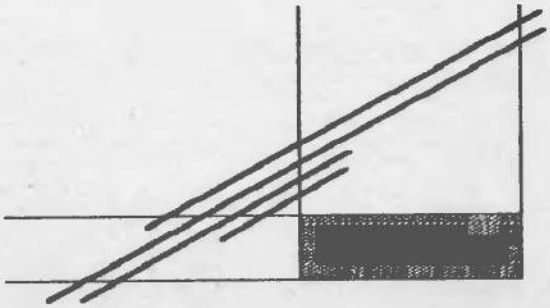


Рис. 25

Если все, что Вам известно о таких линиях - это только координаты начала и конца, то весьма сложно создать алгоритм, по которому программа будет принимать решение о том, проходит или нет линия через экран. К счастью, есть вариант обхода проблемы. Все, что для этого нужно, это рассчитать X1'', Y1'' и

X2'',Y2'' по вышеописанной методике и, если после этого, какая-либо из точек по-прежнему выходит за пределы экрана, то и вся линия не лежит в области экрана и ие должна изображаться.

Окружности.

Окружности могут изображаться, как ряд очень малых отрезков прямых, что дает впечатление непрерывной замкнутой кривой линии. Другими словами, вместо изображения окружности мы можем рисовать правильный многоугольник с большим числом очень малых сторон. Чем больше количество сторон, тем больше наш многоугольник похож на настоящую окружность. В идеале количество сторон следует принимать таким, чтобы оно было кратно четырем, тогда результирующее изображение получится симметричным.

В чисто практических целях можно приближенно определить количество сторон многоугольника для окружности заданного радиуса с помощью эмпирической функции

n = PI*SQR(R), где R - радиус.

Для той разрешающей способности, которую имеет "Спектрум", эта формула работает очень хорошо:

R	n	R	n
10	12	100	32
20	16	160	40
30	16	200	44
40	20	256	52

Если же теперь мы подвергнем каждый отрезок процедуре "клиппирования", которая была описана выше, и не будем рисовать те отрезки или их части, которые не лежат в области экрана, то в результате получим клиппированную окружность. Только та часть окружности, которая проходит через экран, будет изображена, все

же остальное будет проигнорировано.

Развивая этот принцип далее, мы сможем применить ту же идею и для изображения других типов кривых, а не только окружностей, например для эллипсов (Рис. 26).

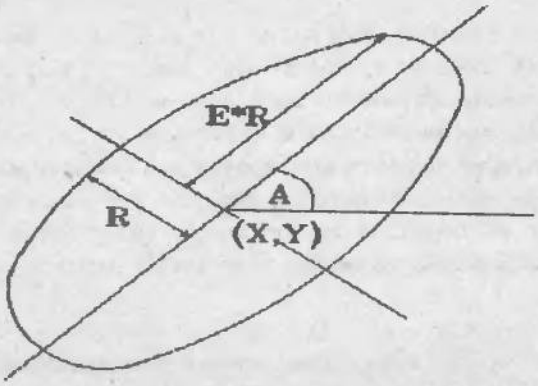


Рис. 26

X и Y - координаты центра эллипса. R - размер его малой полуоси. E - эксцентриситет, он равен отношению большей полуоси эллипса к меньшей и всегда больше единицы. A - угол в радианах между большей полуосью эллипса и горизонталью.

Ниже мы привели процедуры в машинных кодах, которые реализуют описанные приемы.

```
FN S(X1,Y1,X2,Y2) - клиппированные прямые;
FN C(X,Y,R)       - клиппированные окружности;
FN E(X,Y,R,E,A)   - клиппированные эллипсы.
```

Посмотреть как они работают, Вы можете, набрав небольшую демонстрационную программу на БЕЙСИКе. Вместо многоточия в операторах USR..... следует подставить конкретные адреса, соответствующие точкам входа в процедуры, вычерчивающие клиппированные отрезки, окружности и эллипсы.

```
 10 DEF FN S(A,B,C,D) = USR ........
 20 DEF FN C(X,Y,R) = USR ........
 30 DEF FN E(X,Y,R,E,A) = USR ........
100 FOR K = 0 TO PI STEP PI/20
110 RANDOMIZE FN E(128,88,60,2,K)
120 NEXT K
130 FOR K= 60 TO 120 STEP 20
140 FOR J= 0 TO 4
150 LET A1 = (2*J/5 + 0.5)*PI
160 LET A2 = (2*(J+2)/5 + 0.5)*PI
170 RANDOMIZE FN S(128 + K*COS A1, 88 + K*SIN A1,
    128 + K*COS A2, 88 + K*SIN A2)
180 NEXT J
190 RANDOMIZE FN C(128,88,K)
200 NEXT K
```

В своей работе процедуры активно используют встроенный калькулятор, при этом им необходимы двадцать ячеек внутренней памяти калькулятора. Стандартно калькулятор имеет только шесть ячеек памяти. Дополнительную память необходимо создать в рабочем пространстве, используя команду процессора RST 30H. Кроме этого, нужно установить системную переменную MEM, определяющую положение ячеек памяти калькулятора так, чтобы она указывала на эту дополнительную память. После работы процедур необходимо восстанавливать значение MEM в первоначальное (5C92H), что и сделано в программе перед выходом.

Вот раскладка этих ячеек памяти калькулятора: (первые десять служат для клиппирования отрезков прямых, последующие десять служат для работы с окружностями и эллипсами).

Ячейка	Параметр	Комментарий
M0	P1	Координата X точки начала отрезка минус 127.5, т.е. эта координата "привязана" к центру экрана.
M1	Q1	Координата Y точки начала отрезка минус 87.5

M2	P2	Координата X конца отрезка минус 127.5.
M3	Q2	Координата Y конца отрезка минус 87.5.
M4	S1	Вспомогательный флаг, характеризующий положение точки X1 относительно экрана. Если точка слева от экрана, то S1=-1; если точка справа от экрана, то S1=1; если X1 принадлежит экрану, то S1=0.
M5	T1	Вспомогательный флаг, характеризующий положение точки Y1 относительно экрана. Если точка ниже экрана, то T1=-1; если точка выше экрана, то T1=1; если Y1 принадлежит экрану, то T1=0.
M6	S2	То же, что и S1, но для координаты X2 конца отрезка.
M7	T2	То же, что и T1, но для координаты Y2 конца отрезка.
M8	127.5	Постоянное число - половина ширины экрана.
M9	87.5	Постоянная - половина высоты экрана.
MA	E*R	Размер большей полуоси эллипса.
MB	R	Меньшая полуось эллипса (для окружности - радиус).
MC	COS(A)	Характеризуют наклон большей полуоси эллипса к горизонтали.
MD	SIN(A)	
ME	A1	Текущее значение угла наклона A.
MF	INCR	Шаг, с которым нарашивается текущее значение угла наклона A.
M10	P	Экранная X-координата текущей изображаемой точки эллипса (окружности).
M11	Q	Экранная Y-координата текущей изображаемой точки эллипса (окружности).
M12	X	Экранная X-координата центра эллипса.
M13	Y	Экранная Y-координата центра эллипса.

Если Вы входите в процедуры из БЕЙСИКа, как показано в демонстрационной программе, то передача параметров осуществляется через аргументы пользовательских функций FN (), но если Вы хотите войти в эти процедуры из машинного кода, то надо

самим позаботиться о передаче параметров. В этом случае они должны быть предварительно помещены на вершину стека калькулятора (обязательно в указанном порядке):

FN S(...) - X1, Y1, X2, Y2
FN E(...) - X, Y, R, E, A

Поскольку окружность это частный случаи эллипса, у которого эксцентриситет равен единице, а угол наклона большой полуоси равен нулю, то для изображения окружности из машинного кода можно пользоваться той же точкой входа, что и для эллипса, а на вершине стека калькулятора необходимо предусмотреть последовательность:

X, Y, R, 1, 0

Использованные процедуры.

Процедуры ПЗУ

RST 30H - процедура служит для создания дополнительных ячеек памяти калькулятора. На каждую ячейку выделяется по пять байтов. Суммарное количество байтов, которое нужно выделить для памяти калькулятора устанавливается перед вызовом RST 30H в регистровой паре BC. Важно также иметь ввиду, что после работы этой процедуры регистровая пара DE указывает на первый байт выделенной дополнительной области.

STACK_NUM (33B4H=13236 DEC) - процедура помещает на вершину стека калькулятора интегральное (пятибайтное) число, которое начинается с адреса, на который указывает регистровая пара HL. Применяется для ввода на стек тех параметров, которые пользователь подставил в пользовательской функции FN ().

FP_TO_A (2DD5H=11733 DEC) - процедура снимает с вершины

стека калькулятора действительное число, конвертирует его в целое и переносит в аккумулятор микропроцессора. Если получаемое при этом число больше, чем 255, включается флаг переноса - C. О знаке числа можно судить по состоянию флага нуля - Z. Если число положительное или ноль, флаг Z включен, а если отрицательное - выключен.

PLOT (22DCH = 8924 DEC) - при этой точке входа процедура PLOT снимает со стека два верхних значения и, считая, что это экранные координаты точки Y и X, напечатает точку в данных координатах.

LINE_DRAW (2477H = 9335 DEC) - процедура рисует отрезок прямой от последней изображенной на экране точки к точке, координаты которой отстоят от последней на величину смещения, которое задано на вершине стека калькулятора.

STACK_A (2D28H = 11560 DEC) - процедура переводит целое число, содержащееся в аккумуляторе, в действительное число и помещает его на вершину стека калькулятора.

Процедуры программы.

MAIN - головная процедура, с которой начинается вся работа. Отсюда распределяется работа по прочим процедурам программы.

SET_UP - процедура создает необходимое количество ячеек памяти калькулятора (10 или 20) и помещает на вершину стека калькулятора те параметры, которые были заданы в функции пользователя FN () при вызове программы.

CLIP - головная процедура для изображения одного клиппированного отрезка. Здесь рассчитываются координаты начала и конца отрезка относительно центра экрана (P1, Q1, P2, Q2) и устанавливаются в соответствующие ячейки памяти калькулятора M0...M3.

Здесь же выставляются постоянные в ячейках M8, M9, определяется возможность изображения части отрезка на экране и, если это возможно, отрезок рисуется вызовом процедур PLOT и LINE_DRAW из системного ПЗУ компьютера.

PARAMS - процедура, которая по заданным координатам начала и конца отрезка определяет состояние флагов S1,T1,S2,T2 и заносит их в соответствующие ячейки памяти калькулятора M4...M7.

CONV_X - процедура рассчитывает координаты пересечения нашего отрезка с горизонтальными границами экрана P1',Q1' или P2',Q2'.

CONV_Y - процедура рассчитывает координаты пересечения нашего отрезка с вертикальными границами экрана P1",Q1" или P2",Q2".

EL_MAIN - головная процедура для изображения клиппированного эллипса (или окружности). Здесь инициализируются ячейки памяти калькулятора, связанные с эллипсами и окружностями, производится аппроксимация до N-угольника и организуется цикл изображения сторон N-угольника с помощью процедуры рисования клиппированных отрезков CLIP.

NEXT_POINT - процедура вычисляет экранные координаты X1 или X2 и Y1 или Y2 для текущей точки эллипса по заданным R, E, A, A1. Конечная цель процедуры вычислить:

$$X1(X2) = X+E*R*COS(A1)*COS(A)-R*SIN(A1)*SIN(A)$$
$$Y1(Y2) = Y+E*R*COS(A1)*SIN(A)+R*SIN(A1)*COS(A)$$

Процедура MAIN

```
Код      Метка   Мнемоника      Комментарий (стек калькулятора)

013200   LINE    LD BC,0032  ;32H = 50 DEC. Это подготовка к
                             ;созданию 10 ячеек памяти каль-
```

```
                                ;кулятора (по 5 байтов на каж-
                                ;дую).
CD????          CALL SET_UP     ;Вызов процедуры SET_UP.
CD????          CALL CLIP       ;Вызов процедуры рисования клип-
                                ;пированного отрезка.
1815            JR EXIT         ;Переход на финишные операции.
016400  CIRCLE  LD BC,0064      ;Точка входа для изображения
                                ;окружности. 64H = 100DEC - это
                                ;подготовка к созданию 20-ти
                                ;ячеек памяти калькулятора.
CD????          CALL SET_UP     ;Вызов процедуры SET_UP.
EF              RST 28H         ;Включение калькулятора:
                                ;(X, Y, R).
A1              const_1         ;Т.к. окружность - это эллипс, у
                                ;которого эксцентриситет равен
                                ;единице, то на вершину стека
                                ;заслали число 1.
                                ;(X, Y, R, 1)
A0              const_0         ;Заслали 0 на вершину стека -
                                ;это угол наклона большой полу-
                                ;оси эллипса.
                                ;(X, Y, R, 1, 0).
38              endcalc         ;Выключение калькулятора.
1806            JR ELL_2        ;Переход на рисование эллипса.
016400  ELLIPS  LD BC,0064      ;Точка входа для изображения эл-
                                ;липса. 64H в BC - это подготов-
                                ;ка к созданию 20 ячеек памяти
                                ;калькулятора.
CD????          CALL SET_UP     ;Вызов процедуры SET_UP.
CD????  ELL_2   CALL EL_MAIN    ;Вызов головной процедуры рисо-
                                ;вания эллипса.
21925C  EXIT    LD HL,MEMBOT    ;Восстановление системной пере-
22685C          LD (MEM),HL     ;менной MEM перед выходом из
                                ;программы.
215827          LD HL,2758      ;ВНИМАНИЕ! Многие процедуры
                                ;калькулятора "портят" регистро-
                                ;вую пару H'L' (альтернативную).
                                ;Если в ней не восстановить ис-
```

```
                                ;ходное значение (2758H), то
                                ;возврат в БЕИСИК не получится.
                                ;Все, кто работают с машинным
                                ;кодом, никогда не должны
                                ;забывать об этом.
D9                  EXX         ;Пара HL стала альтернативной.
C9                  RET         ;Конец работы программы и выход
                                ;в БЕИСИК.
```

Процедура SET_UP

Код	Метка	Мнемоника	Комментарии
F7	SET_UP	RST 30	;Создание в рабочей области пространства для размещения дополнительных ячеек памяти калькулятора.
ED53685C		LD (MEM),DE	;В системной переменной MEM устанавливается адрес нового места расположения памяти калькулятора.
2A0B5C		LD HL,(DEFADD)	;HL указывает на первый параметр функции пользователя FN ().
23	SU_LOOP	INC HL	;Пропустили имя этого параметра.
23		INC HL	;Пропустили символ CHR 14, за которым следуют 5 байтов числового значения параметра.
CDB433		CALL STACK_NUM	;Вызов процедуры ПЗУ, которая скопирует значение, на которое указывает HL, на вершину стека калькулятора.
7E		LD A,(HL)	;Проверим, что за байт стоит после принятого параметра.
23		INC HL	;Пропустим этот байт.
FE2C		CP 2C	;Не запятая ли это? Код запятой равен 2C.

```
28F5            JR Z,SU_LOOP;Если это запятая, то значит не
                            ;все параметры приняты из FN ()
                            ;и следует возврат на SU_LOOP.
C9              RET         ;В противном случае возврат в
                            ;вызывающую процедуру.
```

Процедура CLIP

Код	Метка	Мнемоника	Комментарий (стек калькулятора)
EF	CLIP	RST 28	;Включение калькулятора. После
			;Того, как отработала процедура
			;SET_UP, мы имеем на вершине
			;стека X1, Y1, X2, Y2.
34372F		stK_data 87.5	;Заслали на вершину число 87,5.
			;(X1, Y1, X2, Y2, 87.5).
C9		store_M9	;Записали его в ячеику M9.
			;(X1, Y1, X2, Y2, 87.5).
03		subtract	;Вычитание Q2 = Y2-87.5;
			;(X1, Y1, X2, Q2)
C3		store_M3	;Запомнили Q2 в ячейке M3.
02		delete	;Удаление числа с вершины;
			;(X1, Y1, X2)
34377F		stK_data 127.5	;Заслали на вершину число 127,5.
			;(X1, Y1, X2, 127.5).
C8		store_M8	;Записали его в ячеику M8.
			;(X1, Y1, X2, 127.5).
03		subtract	;Вычитание P2 = X2-127.5;
			;(X1, Y1, P2)
C2		store_M2	;Запомнили P2 в ячейке M2.
02		delete	;Удаление числа с вершины;
			;(X1, Y1)
E9		recall_M9	;Вызов числа из M9;
			;(X1, Y1, 87.5)
03		subtract	;(X1, Q1)
C1		store_M1	;Запись в ячейку M1;
02		delete	;(X1).

```
E8              recall_M8   ;Вызов числа из M8;
                            ;(X1, 127.5).
03              subtract    ;(P1) .
C0              store_M0    ;Запись в ячейку M0.
02              delete      ;Стек пуст.
38              endcalc     ;Выход из калькулятора.
CD????          CALL PARAMS ;Устанавливаем параметры
                            ;S1, T1, S2, T2.
EF              RST 28      ;Включение калькулятора.
E4              recall_M4   ;На вершине: S1.
E6              recall_M6   ;На вершине: S1, S2.
04              multiply    ;На вершине: S1*S2.
A1              const_one   ;На вершине: S1*S2, 1.
03              subtract    ;На вершине: S1*S2-1.
E5              recall_M5   ;(S1*S2-1, T1)
E7              recall_M7   ;(S1*S2-1, T1, T2)
04              multiply    ;(S1*S2-1, T1*T2)
A1              const_one   ;(S1*S2-1, T1*T2, 1)
03              subtract    ;(S1*S2-1, T1*T2-1)
04              multiply
38              endcalc     ;Выключение калькулятора.
CDD52D          CALL FP_TO_A ;Результат расчета комплекса
                            ;(S1*S2-1)*(T1*T2-1) помещается
                            ;в аккумулятор.
C8              RET Z       ;Этот комплекс не может иметь
                            ;положительного значения вообще
                            ;никогда и потому флаг Z может
                            ;включиться только если он равен
                            ;нулю, а это бывает тогда и
                            ;только тогда, когда либо S1=S2,
                            ;либо T1=T2. В этих случаях весь
                            ;отрезок лежит слева или справа
                            ;или сверху или снизу от экрана.
                            ;В этом случае его рисовать не
                            ;надо и выполняется возврат в
                            ;вызывающую программу.
EF              RST 28      ;Включение калькулятора.
E4              recall_M4   ;(S1)
```

```
30                  eq_zero       ;(S1=0?)
0010          Jump_true, CLIP_1   ;Переход на CLIP_1, если S1=0.
E4                  recall M4     ;(S1)
38                  endcalc       ;Выключение калькулятора, на
                                  ;вершине стека оставили S1.
CD????              CALL CONV_X   ;Расчет Q1', P1'.
EF                  RST 28        ;Включение калькулятора. На вер-
                                  ;шине стека Q1', P1'.
C0                  store_M0      ;P1' запоминается в M0 вместо P1
02                  delete
C1                  store_M1      ;Q1' запоминается в M1 вместо Q1
02                  delete        ;Стек пуст.
38                  endcalc       ;Выключение калькулятора.
CD????              CALL PARAMS   ;Перейдя к новым значениям коор-
                                  ;динат начала отрезка, мы должны
                                  ;снова пересчитать параметры S,T
E4                  RST 28        ;Включение калькулятора.
E5          CLIP_1  recall_M5     ;T1'.
30                  eq_zero       ;T1'=0?
000B          Jump_true, CLIP_2   ;Переход на CLIP_2, если T1'=0.
E5                  recall_M5     ;T1'.
38                  endcalc       ;T1'.
CD????              CALL CONV_Y   ;Расчет Q1", P1".
EF                  RST 28        ;На стеке: Q1", P1".
C0                  store_M0      ;В M0 теперь P1".
02                  delete        ;На стеке: Q1".
C1                  store_M1      ;В M1 теперь Q1".
02                  delete        ;Стек пуст.
E6          CLIP_2  recall_M6     ;S2.
30                  eq_zero       ;S2=0?
0010          Jump_true, CLIP_3   ;Переход, если S2=0.
E6                  recall_M6     ;S2.
38                  endcalc       ;S2.
CD????              CALL CONV_X   ;Расчет Q2', P2'.
EF                  RST 28        ;На стеке Q2', P2'.
C2                  store_M2      ;В M2 теперь P2'.
02                  delete        ;На стеке: Q2'.
C3                  store_M3      ;В M3 теперь Q2'.
```

```
02                  delete          ;Стек пуст.
38                  endcalc         ;
CD????              CALL PARAMS     ;Пересчет параметров S2' и T2'.
EF                  RST 28
E7          CLIP_3  recall_M7       ;T2'.
30                  eq_zero         ;T2'=0?
000B            Jump_true,CLIP_4    ;Переход, если T2'=0.
E7                  recall_M7       ;T2'.
38                  endcalc         ;T2'.
CD????              CALL CONV_Y     ;Поиск Q2", P2".
EF                  RST 28          ;Q2",P2".
C2                  store_M2        ;В M2 теперь P2".
02                  delete          ;Q2".
C3                  store_M3        ;В M3 - Q2".
02                  delete          ;Стек пуст.
38          CLIP_4  endcalc
CD2580              CALL PARAMS     ;Пересчет S1",S2",T1",T2".
EF                  RST 28          ;Включение калькулятора.
E4                  recall_M4       ;S1".
E5                  recall_M5       ;S1",T1".
07                  or              ;S1" OR T1".
E6                  recall_M6       ;S1" OR T1",S2".
07                  or              ;S1" OR T1" OR S2".
E7                  recall_M7       ;S1" OR T1" OR S2", T2".
07                  or              ;S1" OR T1" OR S2" OR T2".
38                  endcalc
CDD52D              CALL FP_TO_A;В аккумулятор пересылается
                                    ;результат логической операции
                                    ;S1" OR T1" OR S2" OR T2".
A7                  AND A
C0                  RET NZ          ;Если флаг нуля не включен,
                                    ;значит никакая часть отрезка не
                                    ;принадлежит экрану и следует
                                    ;возврат в вызывающую процедуру.
EF                  RST 28          ;Включение калькулятора.
E2                  recall_M2       ;Вызов P2".
E8                  recall_M8       ;P2",127.5.
A2                  const_half      ;P2", 127.5, 0.5.
```

```
0F              add             ;P2", 128.
C8              store_M8        ;Теперь в M8 число 128.
0F              add             ;P2"+128.
27              int             ;X2".
C2              store_M2        ;Теперь в M2 координата X2".
E0              recall_M0       ;X2",P1".
E8              recall_M8       ;X2", P1", 128.
0F              add             ;X2", P1"+128.
27              int             ;X2", X1".
C0              store M0        ;Теперь в M1 координиата X1".
03              subtract        ;X2"-X1".
E3              recall_M3       ;X2"-X1", Q2"
E9              recall_M9       ;X2"-X1", Q2", 87.5
A2              const_half      ;X2"-X1", Q2", 87.5, 0.5
0F              add             ;X2"-X1", Q2", 88
C9              store_M9        ;В M9 число 88.
0F              add             ;X2"-X1", Q2"+88
27              int             ;X2"-X1", Y2"
C3              store_M3        ;В M3 координата Y2".
E1              recall_M1       ;X2"-X1", Y2", Q1"
E9              recall_M9       ;X2"-X1", Y2", Q1", 88
0F              add             ;X2"-X1", Y2", Q1"+88
27              int             ;X2"-X1", Y2", Y1"
C1              store_M1        ;В M1 координата Y1".
03              subtract        ;X2"-X1", Y2"-Y1"
E0              recall_M0       ;X2"-X1", Y2"-Y1", X1"
E1              recall_M1       ;X2"-X1", Y2"-Y1", X1", Y1"
38              endcalc         ;У нас на вершине стека кальку-
                                ;лятора получилась следующая
                                ;конфигурация: верхние два
                                ;вложения выражают координаты
                                ;точки начала экранной части
                                ;отрезка, а нижние два вложения
                                ;выражают "смещение" координаты
                                ;конца клиппированного отрезка
                                ;относительно его начала.
CDDC22          CALL PLOT       ;Вызов процедуры ПЗУ, которая
                                ;напечатает точку с координатами
```

```
                                ;X1",Y1" и снимет два верхних
                                ;вложения со стека калькулятора.
C37724              JP LINE_DRAW;Переходом на процедуру ПЗУ
                                ;LINE_DRAW завершается эта об-
                                ;ширная процедура. LINE_DRAW
                                ;вызовет DRAW_LINE (24B7H), на-
                                ;рисует на экране отрезок и вы-
                                ;полнит завершающий возврат в
                                ;вызывающую процедуру.
```

Процедура PARAMS

```
Код       Метка     Мнемоника     Комментарий (стек калькулятора)

EF        PARAMS    RST 28        ;Включение калькулятора.
E0                  recall_M0     ;X1
31                  duplicate     ;X1, X1
2A                  abs           ;X1, ABS(X1)
E8                  recall_M8     ;X1, ABS(X1), 127.5
03                  subtract      ;X1, ABS(X1) - 127.5
37                  gt_zero       ;X1, ABS(X1)>127.5?
0003            jump_true,PAR_1   ;X1, переход, если ABS(X1)>127.5
02                  delete        ;
A0                  const_zero    ;0
29        PAR_1     sgn           ;S1
C4                  store_M4      ;В M4 хранится S1.
02                  delete        ;
E1                  recall_M1     ;Y1
31                  duplicate     ;Y1, Y1
2A                  abs           ;Y1, ABS(Y1)
E9                  recall_M9     ;Y1, ABS(Y1),  87.5
03                  subtract      ;Y1, ABS(Y1) - 87.5
37                  gt_zero       ;Y1, ABS(Y1)>87.5?
0003            jump_true,PAR_2   ;Y1, переход, если ABS(Y1)>87.5
02                  delete        ;
A0                  const_zero    ;0
```

```
29          PAR_2     sgn             ;T1
C5                    store_M5        ;В M5 хранится T1.
02                    delete          ;
E2                    recall_M2       ;X2
31                    duplicate       ;X2, X2
2A                    abs             ;X2, ABS(X2)
E8                    recall_M8       ;X2, ABS(X2), 127.5
03                    subtract        ;X2, ABS(X2) - 127.5
37                    gt_zero         ;X2, ABS(X2)>127.5?
0003              jump_true,PAR_3     ;X2, переход, если ABS(X2)>127.5
02                    delete          ;
A0                    const_zero      ;0
29          PAR_3     sgn             ;S2
C6                    store_M6        ;В M6 хранится S2.
02                    delete          ;
E3                    recall_M3       ;Y2
31                    duplicate       ;Y2, Y2
2A                    abs             ;Y2, ABS(Y2)
E9                    recall_M9       ;Y2, ABS(Y2), 87.5
03                    subtract        ;Y2, ABS(Y2) - 87.5
37                    gt_zero         ;Y2, ABS(Y2)>87.5?
0003              jump_true,PAR_4     ;Y2, переход, если ABS(Y2)>87.5
02                    delete          ;
A0                    const_zero      ;0
29          PAR_4     sgn             ;T2
C7                    store_M7        ;В M7 хранится T2.
02                    delete          ;
38                    endcalc         ;
C9                    RET             ;Выход из процедуры.
```

Процедура CONV_X

Код	Метка	Мнемоника	Комментарий (стек калькулятора)
EF	CONV_X	RST 28	;Включение калькулятора.
			;На стеке: S

```
E8              recall_M8   ;S, 127.5
04              multiply    ;127.5*S
31              duplicate   ;127.5*S, 127.5*S
E0              recall_M0   ;127.5*S, 127.5*S, X1
03              subtract    ;127.5*S, 127.5*S - X1
E3              recall_M3   ;127.5*S, 127.5*S - X1, Y2
E1              recall_M1   ;127.5*S, 127.5*S - X1, Y2, Y1
03              subtract    ;127.5*S, 127.5*S-X1, Y2-Y1
04              multiply    ;127.5*S,(127.5*S-X1)*(Y2-Y1)
E2              recall_M2   ;127.5*S,(127.5*S-X1)*(Y2-Y1),X2
E0              recall_M0   ;127.5*S, (127.5*S-X1)*(Y2-Y1),
                            ;X2, X1
03              subtract    ;127.5*S, (127.5*S-X1)*(Y2-Y1),
                            ;X2 - X1
05              divide      ;127.5*S, (127.5*S-X1)*(Y2-Y1)/
                            ;(X2-X1)
E1              recall_M1   ;127.5*S, (127.5*S-X1)*(Y2-Y1)/
                            ;(X2-X1), Y1
02              add         ;127.5*S, Y1+(127.5*S-X1)*(Y2-
                            ;-Y1)/(X2-X1)
01              exchange    ;Y1+(127.5*S-X1)*(Y2-Y1)/(X2-X1)
                            ;,127.5*S
38              endcalc
C9              RET
```

Процедура CONV_Y

Код	Метка	Мнемоника	Комментарий (стек калькулятора)
EF	CONV_Y	RST 28	;Включение калькулятора.
			;На стеке: T
E9		recall_M9	;T, 87.5
04		multiply	;87.5*T
31		duplicate	;87.5*T, 87.5*T
E1		recall_M1	;87.5*T, 87.5*T, Y1
03		subtract	;87.5*T, 87.5*T - Y1

```
E2              recall_M2   ;87.5*T, 87.5*T - Y1, X2
E0              recall_M0   ;87.5*T, 87.5*T - Y1, X2, X1
03              subtract    ;87.5*T, 87.5*T-Y1, X2-X1
04              multiply    ;87.5*T,(87.5*T-Y1)*(X2-X1)
E3              recall_M3   ;87.5*T,(87.5*T-Y1)*(X2-X1),Y2
E1              recall_M1   ;87.5*T,(87.5*T-Y1)*(X2-X1),
                            ;Y2, Y1
03              subtract    ;87.5*T,(87.5*T-Y1)*(X2-X1),
                            ;Y2 - Y1
05              divide      ;87.5*T,(87.5*T-Y1)*(X2-X1)/
                            ;(Y2-Y1)
E0              recall_M0   ;87.5*T,(87.5*T-Y1)*(X2-X1)/
                            ;(Y2-Y1), X1
02              add         ;87.5*T, X1+(87.5*T-Y1)*(X2-
                            ;-X1)/(Y2-Y1)
38              endcalc
C9              RET
```

Процедура EL_MAIN

Код	Метка	Мнемоника	Комментарий (стек калькулятора)
EF	EL_MAIN	RST 28	;Включение калькулятора.
			;На стеке: X, Y, R, E, A
31		duplicate	;X, Y, R, E, A, A
20		cos	;X, Y, R, E, A, COS(A)
CC		store_MC	;Ячейка памяти MC хранит COS(A)
02		delete	;X, Y, R, E, A
1F		sin	;X, Y, R, E, SIN(A)
CD		store_MD	;Ячейка памяти MD хранит SIN(A)
02		delete	;X, Y, R, E
01		exchange	;X, Y, E, R
CB		store_MB	;Ячейка памяти MB хранит R
04		multiply	;X, Y, E*R
CA		store_MA	;Ячейка памяти MA хранит E*R

```
02                delete          ;X, Y
D3                store_M13       ;Ячейка памяти M13 хранит Y
02                delete          ;X
D2                store_M12       ;Ячейка памяти M12 хранит X
02                delete          ;Стек пуст.
A0                const_zero      ;0
CE                store_ME        ;В ячейке ME выставляется ноль.
02                delete          ;Стек пуст.
EB                recall_MB       ;R
2A                abs             ;ABS(R)
31                duplicate       ;ABS(R), ABS(R)
EA                recall_MA       ;ABS(R), ABS(R), E*R
2A                abs             ;ABS(R), ABS(R), ABS(E*R)
03                subtract        ;ABS(R), ABS(R)-ABS(E*R)
37                gt_zero         ;ABS(R), ABS(R)>ABS(E*R)?
0004          jump_true,ELL_1     ;ABS(R)  (Переход, если
                                  ;ABS(R)>ABS(E*R)
02                delete          ;Стек пуст.
EA                recall_MA       ;E*R
2A                abs             ;ABS(E*R)
          ELL_1                   ;На стеке содержится некое число
                                  ;W, которое равно либо ABS(R),
                                  ;либо ABS(E*R) в зависимости от
                                  ;того, какое из них больше.
28                sqr             ;SQR(W)
A3                const_PI/2      ;SQR(W),PI/2
04                multiply        ;SQR(W)*PI/2
38                endcalc         ;SQR(W)*PI/2
34                INC (HL)        ;При выходе из калькулятора
                                  ;регистровая пара HL всегда
                                  ;указывает на первый байт (байт
                                  ;экспоненты) числа, находящегося
                                  ;на вершине стека калькулятора.
                                  ;Увеличение экспоненты на едини-
                                  ;цу, эквивалентно умножению чис-
                                  ;ла на 2. Итак, теперь на стеке:
                                  ;SQR(W)*PI.
                                  ;Мы получили ту эмпирическую
```

```
                                ;Формулу (см. выше), с помощью
                                ;которой можно определить сколь-
                                ;ко отрезков прямых нужно для
                                ;изображения окружности или
                                ;эллипса.
CDD52D          CALL FP_TO_A;Выражение SQR(W)*PI округля-
                                ;ется до ближайшего целого и пе-
                                ;редается в аккумулятор процес-
                                ;сора.
3806            JR C,ELL_FC ;Если включился флаг переноса,
                                ;значит число оказалось больше
                                ;255, а нам столько отрезков не
                                ;нужно.
E6FC            AND FC      ;Два младших бита принудительно
                                ;гасятся для того, чтобы
                                ;результат был кратен четырем.
C604            ADD A,04    ;Округляем результат вверх до
                                ;ближайшего целого, делящегося
                                ;на четыре.
3002          JR NC,ELL_DRAW ;Если <256, то переход.
3EFC    ELL_FC  LD A,FCH    ;Если число отрезков >255, то
                                ;выставляем 252, считая, что нам
                                ;этого достаточно.
F5      ELL_DRAW PUSH AF    ;Запомнили число отрезков на
                                ;машинном стеке.
CD282D          CALL STACK_A;И поместили его на стек кальку-
                                ;лятора.
EF              RST 28      ;Включение калькулятора. На сте-
                                ;ке число N - количество отрез-
                                ;ков, с помощью которых аппрок-
                                ;симируется эллипс или окруж-
                                ;ность.
A3              const_PI/2  ;N, PI/2
38              endcalc     ;N, PI/2 - выход из калькулято-
                                ;ра. Пара HL указывает на байт
                                ;экспоненты числа на вершине
                                ;стека.
3683            LD (HL),83H ;После этого на вершине стека:
```

```
                                ;N, 2*PI
EF                RST 28        ;N, 2*PI
01                exchange      ;2*PI, N
05                divide        ;2*PI/N - получили величину шага
                                ;по углу для построения N-уголь-
                                ;ника.
CF                store_MF      ;Запомнили шаг по углу в ячейке
                                ;памяти калькулятора MF.
02                delete        ;Стек пуст.
38                endcalc       ;
CD????          CALL NXT_POINT  ;Процедура возвращает экранные
                                ;координаты X1 и Y1 последней
                                ;точки эллипса в ячейках памяти
                                ;калькулятора M10 и M11.
C1                POP BC        ;Сняли с машинного стека ранее
                                ;сохраненное там число N и взяли
                                ;его в регистр B для организации
                                ;цикла построения N-угольника.
C5        ELL_LOOP PUSH BC      ;
EF                RST 28        ;
EE                recall_ME     ;A1 - текущий угол.
EF                recall_MF     ;A1, INCR - шаг по углу.
0F                add           ;A1+INCR
CE                store_ME      ;В ячейку ME заносится новое из-
                                ;мененное значение текущего уг-
                                ;ла.
02                delete        ;Стек пуст.
F0                recall_M10    ;X1
F1                recall_M11    ;X1,Y1
CD????          CALL NXT_POINT  ;Теперь эта процедура возвращает
                                ;экранные координаты X2 и Y2
                                ;в ячейках M10 и M11.
EF                RST 28        ;X1, Y1
F0                recall_M10    ;X1, Y1, X2
F1                recall_M11    ;X1, Y1, X2, Y2
CD????            CALL CLIP     ;Вызов процедуры изображения
                                ;отрезка.
C1                POP BC        ;B - счетчик сторон N-угольника.
```

```
10E9            DJNZ ELL_LOOP ;Возврат на повтор, если не все
                              ;стороны нарисованы.
C9                RET         ;Возврат в БЕЙСИК.
```

Процедура NXT_POINT

Код	Метка	Мнемоника	Комментарий (стек калькулятора)
EF	EL_MAIN	RST 28	;Включение калькулятора.
EA		recall_MA	;E*R
EE		recall_ME	;E*R, A1
20		cos	;E*R, COS(A1)
04		multiply	;E*R*COS(A1)
C8		store_M8	;Ячейка памяти M8 временно ис-
			;пользуется не по осиовному наз-
			;начению для хранения числа
			;E*R*COS(A1).
EC		recall_MC	;E*R*COS(A1),COS(A)
04		multiply	;E*R*COS(A1)*COS(A)
EB		recall_MB	;E*R*COS(A1)*COS(A),R
EE		recall_ME	;E*R*COS(A1)*COS(A),R,A1
1F		sin	;E*R*COS(A1)*COS(A),R,SIN(A1)
04		multiply	;E*R*COS(A1)*COS(A),R*SIN(A1)
C9		store_M9	;В M9 запоминается R*SIN(A1)
ED		recall_MD	;E*R*COS(A1)*COS(A),R*SIN(A1),
			;SIN(A)
04		multiply	;E*R*COS(A1)*COS(A),R*SIN(A1)*
			;*SIN(A)
03		subtract	;E*R*COS(A1)*COS(A) - R*SIN(A1)*
			;*SIN(A)
F2		recall_M12	;E*R*COS(A1)*COS(A) - R*SIN(A1)*
			;*SIN(A),X
0F		add	;X + E*R*COS(A1)*COS(A) -
			;-R*SIN(A1)*SIN(A)
D0		store_M10	;Полученное значение равно гори-
			;зонтальной экраннои координате
			;вершины многоугольника. Она

```
                                ;сохраняется в M10.
02              delete          ;Стек пуст.
E8              recall_M8       ;E*R*COS(A1)
ED              recall_MD       ;E*R*COS(A1),SIN(A)
04              multiply        ;E*R*COS(A1)*SIN(A)
E9              recall_M9       ;E*R*COS(A1)*SIN(A),R*SIN(A1)
EC              recall_MC       ;E*R*COS(A1)*SIN(A),R*SIN(A1),
                                ;COS(A)
04              multiply        ;E*R*COS(A1)*SIN(A),R*SIN(A1)*
                                ;*COS(A)
0F              add             ;E*R*COS(A1)*SIN(A)+R*SIN(A1)*
                                ;*COS(A)
F3              recall_M13      ;E*R*COS(A1)*SIN(A)+R*SIN(A1)*
                                ;*COS(A), Y
0F              add             ;Y+E*R*COS(A1)*SIN(A)+R*SIN(A1)*
                                ;*COS(A)
D1              store_M11       ;Полученное значение равно вер-
                                ;тикальной экранной координате
                                ;вершины многоугольника. Она
                                ;сохраняется в M11.
02              delete          ;Стек пуст.
38              endcalc
C9              RET             ;Возврат в вызывающую процедуру.
```

Специально для тех наших читателей, кто еще не освоил программирование в машинных кодах и кодах встроенного калькулятора мы привели реализацию того же самого алгоритма на БЕЙСИКе.

```
1 DEF FN x(s,K,J,m,n) = J+(127.5*s-K)*(n-J)/(m-K): REM эта
  функция аналогична процедуре CONV_X.
2 DEF FN y(t,K,J,m,n) = K+(87.5*t-J)*(m-K)/(n-J): REM эта
  функция аналогична процедуре CONV_Y.
3 DEF FN a(x,e,r,a,b) = x+e*r*COS(b)*COS(a)-r*SIN(b)*SIN(a):
  REM эта функция применяется при построении окружностей и
  эллипсов и является аналогом первой половины процедуры
  NXT_POINT.
```

```
  4 DEF FN b(y,e,r,a,b) = y+e*r*COS(b)*SIN(a)+r*SIN(b)*COS(a):
    REM Функция является аналогом второй половины процедуры
    NXT_POINT.
 10 CLS
 20 PRINT AT 7,8; "Press n for liNe"
 30 PRINT AT 9,8; "Press c for Circle"
 40 PRINT AT 11,8; "Press e for Ellipse"
 50 LET a$ = INKEY$
 60 IF a$ = "n" THEN GO TO 100
 70 IF a$ = "c" OR a$ ="e" THEN GO TO 200
 80 GO TO 50
 97 REM
 98 REM**************************************************
 99 REM
100 CLS
110 INPUT "X1=?";x1
120 INPUT "Y1=?";y1
130 INPUT "X2=?";x2
140 INPUT "Y2=?";y2
150 GO SUB 1000
160 STOP
197 REM
198 REM**************************************************
199 REM
200 CLS
210 INPUT "R=?";r
220 INPUT "X=?";x
230 INPUT "Y=?";y
240 IF a$ ="c" THEN LET e=1: LET a=0: GO TO 270
250 INPUT "E=?";e
260 INPUT "A=?";a
270 GO SUB 2000
280 STOP
997 REM
998 REM Подпрограмма аналогична процедуре CLIP
999 REM
1000 LET x1 = x1-127.5: LET y1 = y1-87.5:
     LET x2 = x2-127.5: LET y2 = y2-87.5
```

```
1010 GO SUB 1500
1020 IF (s1*s2-1)*(t1*t2-1)=0 THEN RETURN
1030 IF s1=0 THEN GO TO 1060
1040 LET y1=FN x(s1,x1,y1,x2,y2): LET x1=127.5*s1
1050 GO SUB 1500
1060 IF t1=0 THEN GO TO 1080
1070 LET x1=FN y(t1,x1,y1,x2,y2): LET y1=87.5*t1
1080 IF s2=0 THEN GO TO 1110
1090 LET y2=FN x(s2,x1,y1,x2,y2): LET x2=127.5*s2
1100 GO SUB 1500
1110 IF t2=0 THEN GO TO 1130
1120 LET x2=FN y(t2,x1,y1,x2,y2): LET y2=87.5*t2
1130 GO SUB 1500
1140 IF s1<>0 OR t1<>0 OR s2<>0 OR t2<>0 THEN RETURN
1150 LET x2=INT(x2+128): LET x1=INT(x1+128):
     LET y2=INT(y2+88): LET y1=INT(y1+88)
1160 PLOT x1,y1
1170 DRAW x2-x1,y2-y1
1180 RETURN

1497 REM
1498 REM Подпрограмма аналогична процедуре PARAMS
1499 REM
1500 LET s1=0: LET s2=0: LET t1=0: LET t2=0
1510 IF x1 <-127.5 THEN LET s1 =-1
1520 IF x1 > 127.5 THEN LET s1 = 1
1530 IF x2 <-127.5 THEN LET s2 =-1
1540 IF x2 > 127.5 THEN LET s2 = 1
1550 IF y1 <-87.5 THEN LET t1 =-1
1560 IF y1 > 87.5 THEN LET t1 = 1
1570 IF y2 <-87.5 THEN LET t2 =-1
1580 IF y2 > 127.5 THEN LET t2 = 1
1590 RETURN


1997 REM
1998 REM Подпрограмма аналогична процедуре EL_MAIN
1999 REM
```

```
2000 LET n = PI*SQR(ABS(e*r))
2010 LET n = 4+(4*(INT(n/4)))
2020 IF n>252 THEN LET n=252
2030 LET da=2*PI/n
2035 LET a1=0
2040 FOR z=1 TO n
2050 LET x1 = FN a(x,e,r,a,a1)
2060 LET y1 = FN b(y,e,r,a,a1)
2070 LET a1=a1+da
2080 LET x2 = FN a(x,e,r,a,a1)
2090 LET y2 = FN b(y,e,r,a,a1)
2100 GO SUB 1000
2110 NEXT z
2120 RETURN
```

3.2. Масштабирование.

Итак, мы с Вами научились изображать на экране любые отрезки прямых, а не только те, которые полностью на этом экране помещаются. Научившись изображать отрезки, можете считать, что вы научились изображать любые геометрические фигуры или тела, поскольку немного поработав с бумагой в клеточку Вы сможете представить любое тело в виде набора отрезков (правда это может выглядеть грубовато, но это уже дело вкуса, терпения, таланта и зависит от того, сколько оперативной памяти Вы готовы пожертвовать на хранение координат концов своих отрезков и какое Вам нужно быстродействие).

Теперь оказывается, что то, как изображается отрезок на экране, зависит не только от того, какие он имеет номинальные размеры в массиве данных, а еще и от его положения относительно наблюдателя и экрана. Существуют ведь законы проективной связи. Естественно, чем дальше от Вас находится объект, тем меньше он должен выглядеть на экране. Если Вы, играя в игру типа "Звездных войн", подлетите на своем корабле к кораблю противника, то он должен увеличиться в размерах, может быть на нем смогут появиться новые конструкционные детали, может быть даже он станет таким большим, что не впишется в Ваш экран и тогда процедуры

клиппирования изображения "отрежут" ту часть, которая на экран не попадает. Так или иначе, но перед нами встает проблема масштабирования изображения объекта на экране в зависимости от его удаленности от экрана и от точки начала координат, в которой находится наблюдатель.

Эта концепция будет Вам более понятна, если Вы взглянете на Рис. 27.

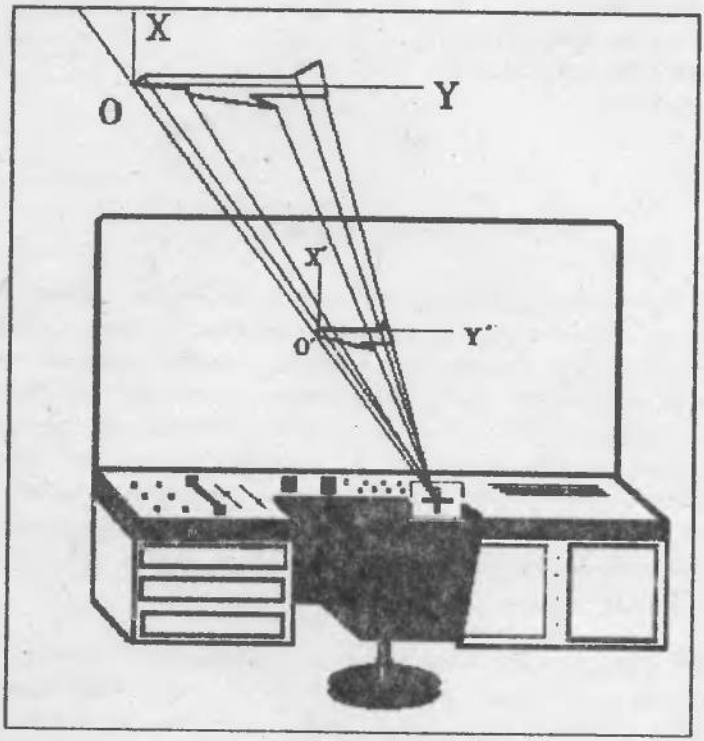


Рис. 27

К счастью, определить экранные координаты трехмерного объекта, удаленного в пространстве от Вас и от проекционного экрана, оказывается очень просто, если предположить, что точка начала координат и положение наблюдателя совпадают (а так оно в большинстве случаев и есть). Формулы для расчета экранных коор-

динат оказываются довольно простыми, известными из элементарной геометрии:

$$X' = X(d/Z)$$
$$Y' = Y(d/Z) \qquad (1)$$

Как видите, масштабирование изображения происходит в соответствии со значением d, которое Вы выбрали сами, см. Рис. 28.

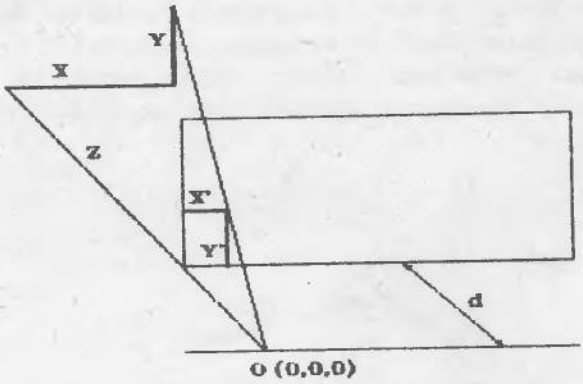


Рис. 28.

Чтобы трансформировать на экран не точечный объект, а объемное тело, достаточно определить экранные координаты всех его вершин и соединить их на экране отрезками прямых.

3.3 Трехмерная векторная графика.

Здесь мы рассмотрим те приемы, которые служат для изображения трехмерных объектов на плоском экране телевизора или монитора.

Все тела, которые окружают нас в реальной жизни, являются трехмерными, т. е. имеют длину, ширину и высоту. Космический

корабль "Таргонов", если Вы его видели, тоже трехмерный, а вот изображение его на Вашем экране всего лишь двумерное, поскольку экран плоский. Так было и так будет по крайней мере до тех пор, пока ученые не изобретут способ подключения к компьютеру мониторов, создающих трехмерные изображения, например с помощью голографической техники. Для нас же это означает то, что нам надо научиться так изображать на плоском экране объемные тела, чтобы создалась иллюзия, что они и на самом деле объемные. Такая техника исполнения изображении и называется трехмерной графикой (3D-графикой). В основу изображения трехмерных тел положено проектирование координат их вершин на плоскость и соединение полученных точек между собой. Существует очень и очень много способов получения проекций. Мы же здесь остановимся только на одном.

Взгляните на Рис. 29. Здесь изображен куб.

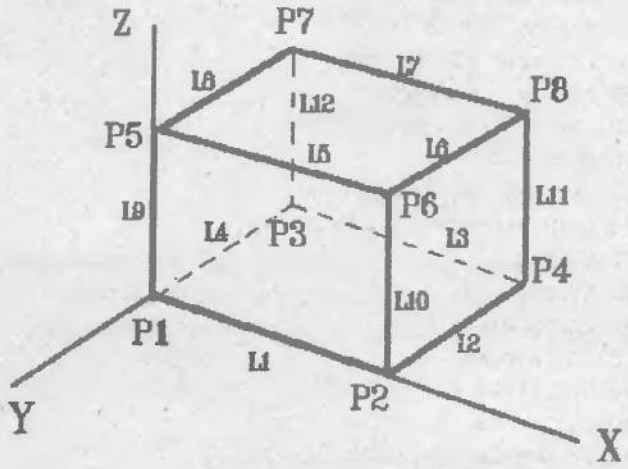


Рис. 29

Куб имеет восемь вершин (P1,..P8) и двенадцать ребер (L1...L12). Ниже приведена программа на БЕЙСИКе, которая нарисует такой куб на экране.

```
10 DIM P(8,3)
20 DIM W(12,2)
```

```
 30 FOR m=0 TO 1
 40 FOR J=0 TO 1
 50 FOR K=0 TO 1
 60 LET P(4*m+2*J+K+1,1)=10*K
 70 LET P(4*m+2*J+K+1,2)=10*J
 80 LET P(4*m+2*J+K+1,3)=10*m
 90 NEXT K
100 NEXT J
110 NEXT m
120 FOR m=1 TO 12
130 FOR J=1 TO 2
140 READ W(m,J)
150 NEXT J
160 NEXT m
170 DATA 1,2,2,4,4,3,3,1
180 DATA 5,6,6,8,8,7,7,5
190 DATA 1,5,2,6,4,8,3,7
200 FOR m=1 TO 12
210 LET a=1: GO SUB 500
220 LET P1 = 5*P+128
230 LET Q1 = 5*Q+88
240 PLOT P,Q
250 LET a=2: GO SUB 500
260 DRAW 5*P+128-P1, 5*Q+88-Q1
270 NEXT m
280 STOP
500 LET a=W(m,a)
510 LET X=P(a,1)
520 LET Y=P(a,2)
530 LET Z=P(a,3)
540 RANDOMIZE USR XXXXX
550 RETURN
```

Не обращайте пока внимания на оператор RANDOMIZE USR в строке 540. Сейчас пока он нас не интересует, и мы к нему еще вернемся.

В строке 10 задается массив P(), в котором хранятся коор-

динаты вершин. Его размерность 8 X 3, поскольку куб имеет 8 вершин, а каждая вершина имеет три координаты. В строке 20 задается массив W(), в котором хранятся ребра. Его размерность 12 X 2, поскольку у куба 12 ребер и каждое соединяет по две вершины. Мы здесь экономим память, поскольку не храним в этом массиве координаты концов ребер (потребовался бы массив вдвое больше), а храним только номера соединяемых вершин, благо их координаты уже известны из массива P(). В строках 30...110 инициализируется массив координат вершин P(), а в строках 120... 190 - массив ребер W(). Здесь приходится использовать массив DATA, поскольку нет чисто вычислительного пути сделать так, чтобы программа сама принимала решение о том, какие вершины между собой соединены, а какие - нет.

Если Вы замените строки 60...80 так, чтобы они заканчивались на 8*K, 10*J, 12*M, то получите уже не куб, а прямоугольный параллелепипед.

Теперь вернемся опять к основному нашему вопросу - как же получается, что объемный куб у нас изображен на плоском экране? Обратите, например, внимание на то, что ребро L7 у нас на экране соединено с вершинами P7 и P8 несмотря на то, что это не сам куб, а только его проекция на плоскость. Отсюда следует генеральный вывод: если вершины тела соединены между собой отрезками прямых в пространстве, то проекции этих вершин на плоскость тоже должны быть соединены на экране. Вывод в общем-то совершенно очевидный, но пусть Вас не вводит эта очевидность в заблуждение, ведь именно в этом и состоит вся концепция изображения пространственных тел на плоском экране. По пространственным координатам вершин найдите координаты их проекций на плоскость, пользуясь математическим аппаратом из геометрии и тригонометрии и соедините их между собой отрезками прямых, пользуясь процедурой DRAW_LINE или с помощью технологии построения клиппированных отрезков, описанной выше - и все.

Остался открытым только один вопрос - а как найти плоские координаты проекции вершины тела на плоскость? Это зависит от того, какой метод проектирования мы примем. Как мы уже сказали,

существует очень и очень много разных видов проекции и формулы пересчета координат зависят от избранного вида. Очень хорошо для этой задачи подходит так называемая изометрическая проекция. Она характеризуется тем, что оси X, Y и Z располагают на экране под равными углами - 120 градусов (2*PI/3) - см. Рис. 29а.

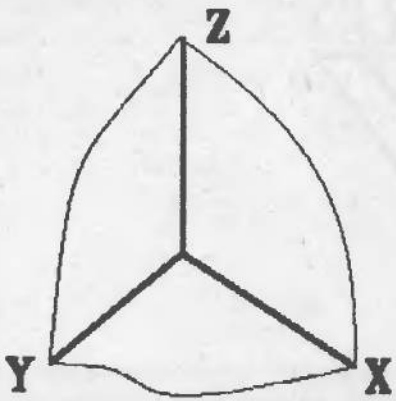


Рис. 29а.

Формулы пересчета оказываются для этого вида проекции очень простыми. Так, если вершина тела имела пространственные координаты X, Y, Z, то координаты ее проекции на плоскость P и Q будут следующими:

```
LET P=SQR(3)*(Y-X)/2                (2)
LET Q=Z-(Y+X)/2
```

Итак, все оказалось просто и решается даже в БЕЙСИКе. Возьмите миллиметровую бумагу. Вычертите три вида своего объекта. На Рис. 30 мы взяли для примера полицейский корабль "Viper" из игры "ELITE". Пронумеруйте вершины, задайте трехмерные координаты каждой вершины и заполните массив вершин. Пронумеруйте ребра и заполните массив ребер, указав какую пару вершин каждое ребро соединяет. А теперь можете, пользуясь формулами проективной связи (2) рассчитать плоские координаты каждой из вершин.

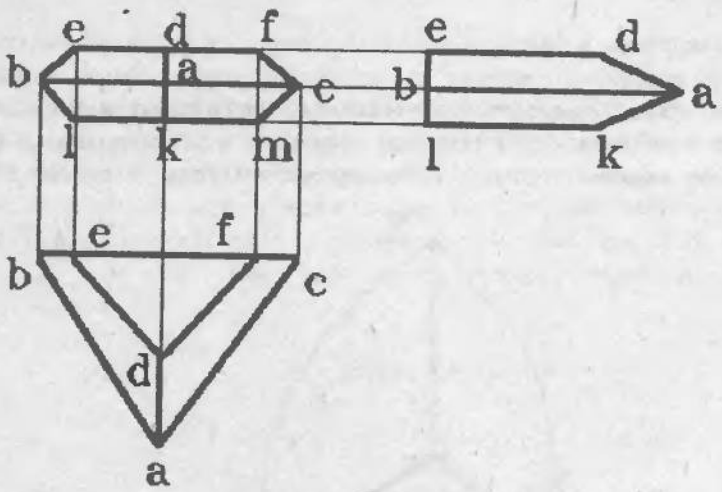


Рис. 30

Теперь, если в Вашей программе предполагается приближение или удаление от объекта, Вы вспомните формулы (1) из предыдущего раздела и пересчитаете плоские координаты объекта в экранные, учитывая удаленность объекта от наблюдателя. Если при этом какие-то вершины окажутся вне пределов экрана, Вам на помощь придет технология клиппирования отрезков.

А теперь вернемся к нашей БЕЙСИК-программе для изображения прямоугольного параллелепипеда. Там не решен только один вопрос - нет формул проективной связи (2), но там есть вызов подпрограммы пользователя RANDOMISE USR XXXXX. Вот этим-то и занимается эта подпрограмма. Конечно, можно было бы расчет этот сделать и в БЕЙСИКе, но БЕЙСИК не слишком быстро занимается математическими расчетами, поэтому эту подпрограмму мы оформили в машинный код с привлечением кодов встроенного калькулятора.

Структурно подпрограмма состоит из трех частей. Первая часть SEARCH_VAR нужна для того, чтобы была связь с головной БЕЙСИК-программой и она служит для того, чтобы найти в области переменных БЕЙСИКа исходные координаты вершины. Вторая часть - STK_ZYX - передает найденные координаты в калькулятор, а третья часть - TRANSFORM занимается непосредственно преобразовани-

ем координат по формулам (2).

Увязка БЕЙСИКа с машинным кодом и передача параметров относится к задачам первостепенной важности. До сих пор мы передавали параметры через FN () или через выделенные для этой цели ячейки памяти. Нижеприведенный алгоритм явится для Вас отличной школой того, как это делается через область программных переменных, поскольку построен весьма рационально.

Процедура SEARCH_VAR

Код	Метка	Мнемоника	Комментарий
2A4B5C	SEARCH_VAR	LD HL,(VARS)	;HL указывает на область прог-
			;раммных переменных.
7E	S_V_LOOP	LD A,(HL)	;Очередной байт из области пе-
			;ременных.
E67F		AND 7F	;Гасим старший бит.
37		SCF	;Включение флага переноса.
C8		RET Z	;Выход с включенным флагом пе-
			;реноса, если найден код 80H,
			;т.е. достигнут конец области
			;переменных, а переменная не
			;найдена.
B9		CP C	;Сравнение найденной переменной
			;с заданной в регистре C.
C8		RET Z	;Возврат с выключенным флагом
			;переноса, если нужная перемен-
			;ная найдена.
C5		PUSH BC	;Запомнили имя исковой перемен-
			;ной. BC необходимо сохранить
			;от порчи при работе вызываемой
			;процедуры NEXT_ONE.
CDB819		CALL 19B8H	;Вызов процедуры системного ПЗУ
			;NEXT_ONE, которая найдет сле-
			;дующую переменную в области
			;VARS и выставит ее адрес в DE.

```
EB              EX DE.HL        ;Перенос этого адреса в HL.
C1              POP BC          ;Восстановили имя переменной.
18F1            JR S_V_LOOP     ;Возврат назад для продолжения
                                ;поиска.
```

Процедура STK_ZYX
~~~~~~~~~~~~~~~~~

Код	Метка	Мнемоника	Комментарии
0E7A	STK_ZYX	LD C.7AH	;7AH = 122 DEC - это код симво-
			;ла z.
CD????		CALL GET_VAR	;Значение переменной z перено-
			;сится на стек калькулятора.
0E79	STK_YX	LD C.79H	;79H = 121 DEC - код "y".
CD????		CALL GET_VAR	;Значение переменной y перено-
			;сится на стек калькулятора.
0E78		LD C.78H	;78H = 120 DEC - код "x".
CD????	GET_VAR	CALL SEARCH_VAR	;Поиск переменной в области
			;программных переменных БЕЙСИКа
DA2E1C		JP C.1C2E	;Если при выходе из SEARCH_VAR
			;включен флаг переноса, то зна-
			;чит такая переменная не наиде-
			;на и следует переход в ПЗУ на
			;процедуру REPORT_2, которая
			;выдаст сообщение об ошибке
			;"Variable not found".
23		INC HL	;Теперь HL указывает не на имя,
			;а на значение переменной.
C3B433		JP 33B4H	;Переход на процедуру ПЗУ
			;STACK_NUM, которая перенесет
			;число, на которое указывает
			;регистровая пара HL на верши-
			;ну стека калькулятора. Она же
			;выполнит возврат в вызывавшую
			;процедуру.
~~~~~~~~~~~~~~~~~

Процедура TRANSFORM

```
Код       Метка       Мнемоника     Комментарии (стек калькулятора)

CD????    TRANSFORM CALL STK_YX    ;Помещаем Y и X на вершину
                                   ;стека калькулятора.
EF                  RST 28H        ;Включение калькулятора.
03                  subtract       ;Y-X
3440B00003          stk_data_3     ;Y-X,3
28                  sqr            ;Y-X,SQR(3)
04                  multiply       ;SQR(3)*(Y-X)
A2                  const_half     ;SQR(3)*(Y-X),1/2
04                  multiply       ;SQR(3)*(Y-X)/2
38                  endcalc
0E70                LD C,70H       ;70H = 112 DEC - код "P".
CD????            CALL ASSIGN_VAR;Переменной, код которой нахо-
                                   ;дится в регистре C, присваива-
                                   ;ется значение, хранящееся на
                                   ;вершине стека калькулятора.
                                   ;   P = SQR(3)*(Y-X)/2
CD????              CALL STK_ZYX ;Принимаем Z, Y и X на стек
                                   ;калькулятора.
EF                  RST 28H        ;Z, Y, X
0F                  add            ;Z, Y+X
A2                  const_half     ;Z, Y+X, 1/2
04                  multiply       ;Z, (Y+X)/2
03                  subtract       ;Z-(Y+X)/2
38                  endcalc
0E71                LD C,71H       ;71H = 113 DEC - код "Q".
CD????  ASSIGN_VAR CALL SEARCH_VAR
300A                JR NC ASSIGN_VAR_2
C5                  PUSH BC        ;Запомнили код имени созда-
                                   ;ваемой переменной.
010600              LD BC,0006     ;Подготовка к выделению до-
                                   ;полнительных шести ячеек
                                   ;памяти в области VARS.
CD5516              CALL 1655H     ;Вызов процедуры ПЗУ MAKE_ROOM.
```

```
                                    ;которая вставит столько новых
                                    ;ячеек памяти, сколько указано
                                    ;в BC. На место вставки указы-
                                    ;вает HL.
23                  INC HL          ;HL указывает на первый байт
                                    ;выделенного пространства.
C1                  POP BC          ;Вернули со стека код имени
                                    ;создаваемой переменной.
71                  LD (HL),C       ;Поместили его в область VARS.
23    ASSIGN_VAR_2  INC HL          ;HL теперь указывает на место-
                                    ;положение числового значения
                                    ;создаваемой переменной.
E5                  PUSH HL
CDBF35              CALL 35BFH      ;Вызов процедуры ПЗУ STK_PNTRS.
                                    ;По этой процедуре в HL выстав-
                                    ;ляется адрес начала верхнего
                                    ;вложения на калькуляторном
                                    ;стеке.
22655C            LD (STKEND),HL    ;Изменяем значение (STKEND).
D1                  POP DE          ;DE теперь указывает на место-
                                    ;положение числового значения
                                    ;создаваемой переменной.
010500              LD BC,0005      ;Подготовка к копированию
                                    ;пяти байтов из (HL) в (DE).
EDB0                LDIR            ;Перенос значения новой пере-
                                    ;менной со стека калькулятора
                                    ;в область VARS.
C9                  RET
```

3.4 Сокрытие линий невидимого контура.

При работе с трехмерной векторной графикой часто встает одна важная проблема - как изображать невидимые линии трехмерного объекта? Эта задача имеет непосредственное отношение к системам автоматизированного проектирования и наибольшее развитие получила именно в теории этих систем и, надо сказать, для ее реше-

ния привлекают довольно сложный аппарат из той области высшей математики, которая называется аналитической геометрией. Для нас с Вами, поскольку мы занимаемся обычной прикладной графикой, эту задачу можно несколько упростить. На данном этапе нас не интересует, как изобразить невидимые линии - нам просто нужно их НЕ ИЗОБРАЖАТЬ.

Приемов и методов для достижения этой цели немало и мы рассмотрим один из наиболее простых, поддающийся несложной алгоритмизации.

Посмотрите на Рис. 31. На нем изображен некоторый трехмерный ландшафт. Фактически это график функции:

Z = SIN (R)/R ,
где R = SQR (X*X + Y*Y) .

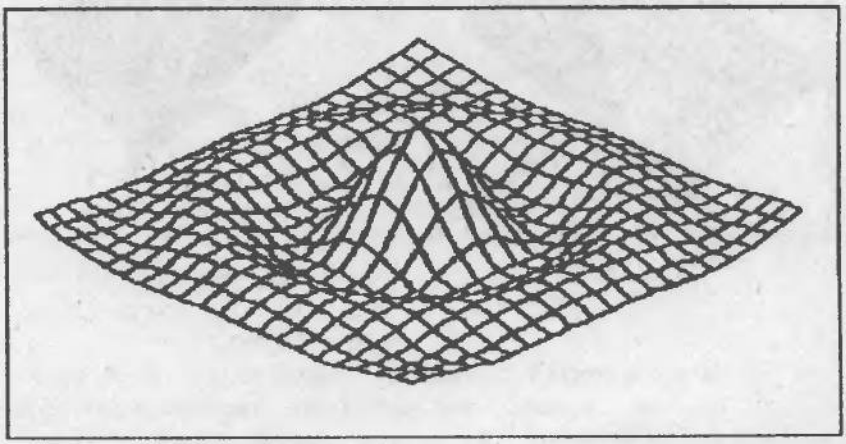


Рис. 31

В своих экспериментах Вы можете изменить эту функцию и поработать с другими. Важно только, чтобы она имела вид Z=f(X,Y) т. е. чтобы была возможность составить однозначный алгоритм для вычисления координаты Z по заданным координатам X и Y. Самое интересное в этом графике - то, что скрытые детали - на самом деле скрыты. Все точки, которые находятся за гребнем или за

вершиной - не показаны.

Алгоритм.

Давайте рассмотрим алгоритм, с помощью которого может быть достигнут желаемый эффект. Во-первых, надо отметить, что наша трехмерная поверхность изображается в два приема. На первом проходе изображаются все линии, параллельные оси X (Рис. 32), а на втором проходе - линии, параллельные оси Y (Рис. 33). Именно благодаря такому порядку изображения кривых и оказывается возможным скрыть невидимые детали.

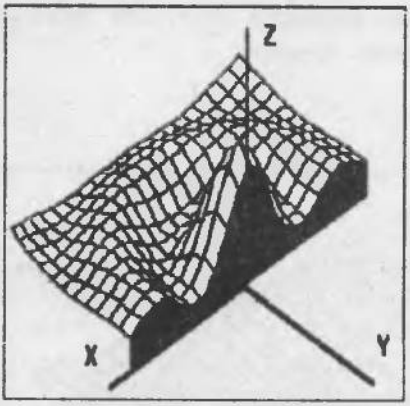


Рис. 32

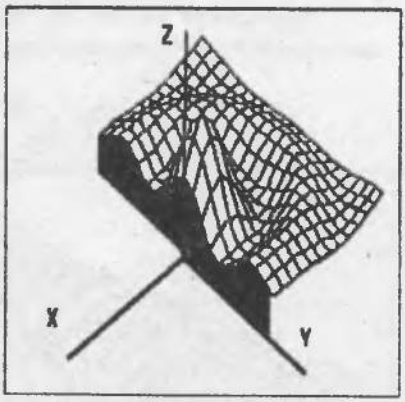


Рис. 33

Линии изображаются с некоторым шагом по X - Xh и по Y - Yh (см. Рис. 34). Конечно, чем мельче шаг, тем детальнее будет проработано изображение, но слишком мельчить тоже не надо - существует некоторый оптимум, который можно установить методом проб и ошибок. Во всяком случае, параметры

$$Xh=(Xmax-Xmin)/20 \quad \text{и}$$
$$Yh=(Ymax-Ymin)/20$$

выглядят достаточно удачными.

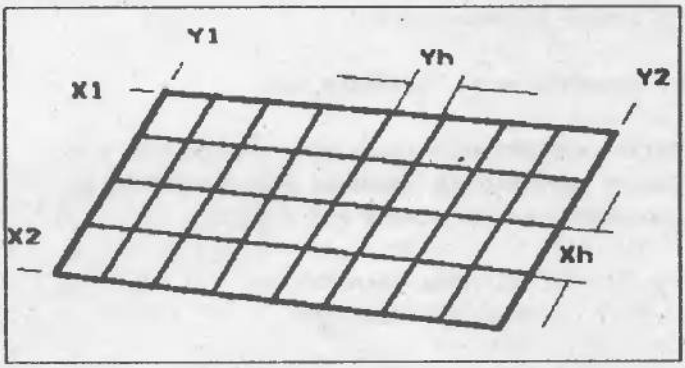


Рис. 34

Если какая-то часть вычерчиваемой в текущий момент линии оказывается за ранее проведенной кривой, то она не изображается и есть достаточно простой прием, который позволяет в программе принять такое решение.

Если мы выстраиваем изображение в виде семейства кривых, начиная от ближайшей к наблюдателю и удаляясь от него назад, (т. е. идем от т. т. X2 и Y2 к т. т. X1 и Y1, как показано на Рис. 34), то фактически те точки текущей линии, которые оказываются на экране ниже, чем точки ранее изображенных линий, и являются невидимыми и должны быть скрыты.

Чтобы решить этот вопрос программно, мы создаем в оперативной памяти буфер размером 256 байтов, а дальше действуем следующим образом. Поскольку экран "Спектрума" имеет в ширину 256 пикселов, то мы будем считать, что он образован как бы из 256-ти узких однопикселных вертикальных столбцов. Каждому столбцу отведем по одной ячейке памяти в нашем буфере и теперь всякий раз, когда будем печатать на экране точку, будем смотреть, что же содержится в буфере для данного столбца. Если то значение, которое есть там - меньше, чем вертикальная координата экрана, в которой мы будем печатать точку, то новая координата запоминается в данном буфере и точка печатается. Если же хранящееся там значение больше, чем текущая вертикальная координата позиции печати, то точка должна быть скрыта и не печатается, а

значение в буфере не изменяется.

Итак, алгоритм имеет следующий вид:

1. Задаем максимальные значения координат X2 и Y2.
2. Задаем минимальные значения координат X1 и Y1.
3. Определяем шаг по осям X и Y - Xh, Yh.

$$Xh=(Xmax-Xmin)/23$$
$$Yh=(Ymax-Ymin)/23$$

Мы специально делили здесь на "некруглое" число 23, а не на 20. Это помогает избежать прохождения через нулевую точку в том случае, если максимальные и минимальные параметры заданы симметрично относительно нуля. Все-таки неуютно себя чувствуешь, когда программа должна посчитать SIN(R)/R, когда R равно нулю. Хотя, в принципе, и этот случай можно было бы предусмотреть. Для тех, кто еще пока не изучал высшую математику, подскажем, что SIN(R)/R приближается к единице, когда R стремится к нулю.

4. Определяем масштаб по осям X и Y.

Для системы координат, показанной на Рис. 31...33 масштаб по X и Y - одинаков. Он зависит от ширины экрана и от угла между осями X, Y и осью Z. Для того, чтобы при любых допустимых значениях X и Y точка умещалась бы на экране, нам необходимо избрать масштаб:

$$(X2-X1)+(Y2-Y1)*255/SQR(3)*2$$

5. Задаем масштаб для оси Z (его можно менять).
6. Создаем буферный массив из 256 элементов, обнуляем их.
7. Начинаем строить семейство кривых, "параллельных" оси X. Организуем цикл по Y от Y2 до Y1 с шагом Yh.

8. Внутри этого цикла организуем цикл от X2 до X1 с шагом по Xh.

9. Внутри этого цикла для текущих значении X и Y определяем Z по заданной формуле функции.

10. Полученный результат для Z умножаем на масштаб, получаем координату Z для графика.

11. Выполняем преобразование систем координат. По трехмерным координатам X, Y, Z находим значения x и y для плоскости экрана.

Формулы для этого преобразования будут зависеть от того, какую проекцию трехмерной системы координат на плоскость Вы изберете. Другими словами, они зависят от того, под какими углами Вы смотрите на трехмерный объект. Для случая, показанного на Рис. 31...33, подойдут формулы:

```
x = SQR(3)*(Y-X)/2 +127
y = Z - (Y+X)/2 + 87
```

12. Мы готовы поставить на экране точку в координатах x, y. Но сначала проверим, что есть в буфере для данной координаты x. Если там значение меньше, чем y, то точку x, y на экране ставим и значение y обновляем в буфере, а если оно больше, то точка - невидима, мы ее не ставим и значение в буфере не обновляем.

13. Вычислив экранные координаты точки x, y, мы готовы соединить ее линией с предыдущей точкой x', y' (если наша точка не первая). Хорошо бы для этого воспользоваться командой БЕЙСИКа DRAW или процедурой изображения отрезков в машинных кодах, но делать этого, к сожалению, нельзя. Причина в том, что этот отрезок (или его часть) может быть невидимым. Значит, надо строить его по точкам и для каждой точки проверять по буферу видима она или нет. Поэтому опять же надо организовать цикл для изображения отрезка по точкам.

14. Теперь надо определиться с параметром этого цикла. Он может изменяться по горизонтали (по x), а может и по вертикали

(по Y). Надо понять, что больше - приращение dx (равное x-x') или dy (равное y-y'). То, которое больше, и следует принять в качестве параметра цикла.

15. Определившись с парметром, организуем цикл и внутри него вычисляем координаты текущих точек, проверяем для них Y, сравнением с буфером и, если точка видима, печатаем ее и обновляем буфер, а если нет, то не печатаем и не обновляем буфер.

Здесь есть маленькая хитрость, которая несколько усложняет жизнь программисту. Дело в том, что эти соединительные отрезки можно проводить слева-направо, а можно и справа налево. В принципе, это все равно, но есть один нюанс. Допустим, мы будем их проводить слева направо. Все будет в порядке, пока нам не придется провести круто падающий отрезок. На одиг шаг по x для него происходит несколько шагов по Y. И бывает так, что одной координате x соответствуют несколько точек Y. Если бы мы рисовали этот отрезок снизу вверх, все было бы в порядке, а при движении сверху вниз (слева-направо) ранее напечатанная точка может "блокировать" печать следующих, забив в буфере свою координату. Поэтому круто падающие отрезки программа должна строить наоборот - справа-налево. Тогда отрезок становится как бы не "падающим", а "восходящим".

16. Соединив две точки на экране, переходим к очередной точке, отстоящей на Xh, и возвращаемся на шаг 8.

17. Посторив кривую, "параллельную" оси X, переходим к следующей, отстоящей от нее на шаг Yh. Возвращаемся на шаг 7.

18. Когда все семейство кривых, "параллельных" оси X построено, половина дела сделана. Теперь надо построить семейство кривых, параллельных оси Y.

19. Для этого сначала переинициализируем буфер, обнулив все его значения, а затем повторим все то же, что мы делали для семейства кривых, идущих вдоль оси X (шаги 7 - 18). Правда,

при этом вместо шагов по X будем делать шаги по Y и наоборот.

Вот практически и весь алгоритм. Его описание выглядит страшнее, чем текст программы на БЕЙСИКе (см. ниже). И это не случайно - ведь БЕЙСИК гораздо лучше подходит для описания алгоритмов и программистских идей, чем нормальный человеческий язык.

Конечно, скорость работы этой программы на БЕЙСИКе оставляет желать лучшего, но для иллюстрации самой концепции он неплох. Вы можете менять максимальные и минимальные значения X и Y. Вы можете менять масштаб по Z. Более того, Вы можете менять и саму функцию, исследуя другие поверхности.

```
 10 LET xmax = 10: LET ymax = 10
 20 LET xmin =-10: LET ymin = -10
 30 LET xh = -(xmax-xmin)/23: LET yh = -(ymax-ymin)/23
 40 LET scale = 255/SQR(3)*2/((xmax-xmin)+(ymax-ymin))
 50 LET zscale = 40
 60 DIM c(256): FOR i=1 TO 256: LET c(i)=0: NEXT i
 70 FOR y=ymax TO ymin STEP yh: LET y1=y*scale
 80 FOR x=xmax TO xmin STEP xh: LET x1=x*scale
 90 GO SUB 5000
100 IF x=xmax THEN GO SUB 8100: NEXT x
110 LET dy=yt-yold: LET dx=xt-xold
120 IF ABS(dy) >= ABS(dx) THEN GO SUB 6000: GO TO 140
130 GO SUB 7000
140 NEXT x
150 NEXT y
155 FOR i=1 TO 256: LET c(i)=0: NEXT i
160 FOR x=xmax TO xmin STEP xh: LET x1=x*scale
170 FOR y=ymax TO ymin STEP yh: LET y1=y*scale
180 GO SUB 5000
190 IF y=ymax THEN GO SUB 8100: NEXT y
200 LET dy=yt-yold: LET dx=xt-xold
210 IF ABS(dy) >= ABS(dx) THEN GO SUB 6000: GO TO 230
220 GO SUB 7000
```

```
 230 NEXT y
 240 NEXT x
 250 STOP
4997 REM
4998 REM*************************************************
4999 REM
5000 LET r = SQR(x*x + y*y): LET z = SIN(r)/r
5010 LET z = z*zscale
5020 LET xt=127+SQR(3)*(y1-x1)/2
5030 LET yt=87+z-(y1+x1)/2
5040 RETURN
5997 REM
5998 REM*************************************************
5999 REM
6000 IF dy<0 THEN GO TO 6100
6010 FOR i=0 TO dy
6020 GO SUB 6500
6030 NEXT i
6040 GO SUB 8100: RETURN
6100 FOR i=dy TO 0
6110 GO SUB 6500
6120 NEXT i
6130 GO SUB 8200: RETURN
6497 REM
6498 REM*************************************************
6499 REM
6500 LET xt=xold+dx/dy*i
6510 LET yt=yold+i
6520 GO SUB 8000: RETURN
6997 REM
6998 REM*************************************************
6999 REM
7000 IF dx<0 THEN GO TO 7100
7010 FOR i=0 TO dx
7020 GO SUB 7500
7030 NEXT i
7040 GO SUB 8100: RETURN
7100 FOR i=dx TO 0
```

```
7110 GO SUB 7500
7120 NEXT i
7130 GO SUB 8200: RETURN
7497 REM
7498 REM ***********************************************
7499 REM
7500 LET xt=xold+1
7510 LET yt=yold+dy/dx*1
7520 GO SUB 8000: RETURN
7997 REM
7998 REM***********************************************
7999 REM
8000 IF yt > c(xt+1) THEN LET c(xt+1)=yt: PLOT xt,yt
8010 RETURN
8097 REM
8098 REM***********************************************
8099 REM
8100 LET xold=xt: LET yold=yt: RETURN
8197 REM
8198 REM***********************************************
8199 REM
8200 LET xold=xold+dx: LET yold=yold+dy: RETURN
```

СПИСОК ПРОГРАММНЫХ ПЕРЕМЕННЫХ

xmax - максимально-допустимое значение координаты X (задается пользователем).

ymax - то же для координаты Y.

xmin - минимально-допустимое значение координаты X (задается пользователем).

ymin - то же для координаты Y.

xh, yh - шаг между узлами сетки.

scale - масштаб по координатам X и Y (зависит от углов в пространстве, под которыми наблюдатель смотрит на трехмерный объект. Рассчитывается исходя из соображении оптимального использования плоскости экрана.

zscale - масштаб по оси Z (задается пользователем "по вкусу".

но так, чтобы изображение по вертикали не вышло за пределы экрана). Возможно и автоматическое определение zscale в программе.

c(256)- буферный массив на 256 элементов.
X1, Y1 - отмасштабированные значения трехмерных координат X и Y.
r - математический комплекс, нужный для вычисления Z.
x, y - текущие трехмерные координаты X и Y в узлах сетки.
xt, yt - текущие экранные координаты (двумерные).
xold, yold - экранные (двумерные) координаты предыдущего узла.
dx, dy - приращения экранных координат на очередном шаге (расстояние на экране между узлами).

Те же, кто предпочтут использовать для программирования машинный код, получат прекрасные результаты, но надо учесть, что программа выполняет большой объем чисто математических вычислений (это вообще характерно для трехмерной векторной графики). Здесь и масштабирование (чтобы график аккуратно занимал плоскость экрана) и пересчет из одной системы координат в другую (из трехмерной системы координат в двумерную систему координат плоскости экрана) и, конечно же, расчет самой функции Z=f(X,Y). Процессор Z-80 не может оперировать с действительными числами, не может выполнять математических расчетов и здесь используют программирование в кодах калькулятора. Ниже приведен пример программы в машинном коде. Подробный комментарий всех входящих процедур явится хорошим пособием для тех, кому необходимо программирование в кодах калькулятора.

Список использованных ячеек памяти калькулятора.

M0	drawX	"Смещение" конца вычерчиваемого отрезка относительно его начала в направлении оси X.
	disPF	"Смещение" конца отрезка в "прямом" (см. Примечание) направлении.
M1	drawY	"Смещение" конца отрезка в направлении оси Y.
	disPT	"Смещение" в "поперечном" (см. Примечание) направлении.
	ratio	Отношение disPF/disPT.

M2	lineX lineF	Координата X очередной точки линии. Координата очередной точки линии в "прямом" направлении.
M3	lineY lineT	Координата Y очередной точки линии. Координата очередной точки в "поперечном" направлении.
M4	plotX plotF	Координата X текущей печатаемой точки. Координата текущей точки линии в "прямом" направлении.
M5	plotY plotT	Координата Y текущей печатаемой точки. Координата текущей точки линии в "поперечном" направлении.
M6	X1	Минимальное значение X (см. рис. 34)
M7	Y1	Минимальное значение Y (см. рис. 34)
M8	X2	Максимальное значение X (см. рис. 34)
M9	Y2	Максимальное значение Y (см. рис. 34)
MA	XH	Шаг в направлении оси X (см. рис. 34).
MB	YH	Шаг в направлении оси Y (см. рис. 34).
MC	Zscale	Масштаб по оси Z (на эту величину умножается полученное значение координаты Z).
MD	XYscale	Масштаб по осям X,Y (половина той величины, на которую умножаются координаты X и Y).
ME	X	Координата X в трехмерном пространстве.
MF	Y	Координата Y в трехмерном пространстве.

Примечание. Каждая кривая состоит из отрезков прямых. А каждый отрезок как-то наклонен к экранным осям X и Y. Если отрезок идет как бы "вдоль" оси X экрана, то прямым направлением считается ось X, а поперечным - ось Y. В то же время, если он ориентирован больше вдоль оси Y экрана, то соответственно наоборот: прямое направление - Y, а поперечное - X.

Вызов подпрограммы.

Здесь реализован вызов подпрограммы и передача в нее параметров через БЕЙСИКовскую функцию пользователя FN G(). Функция пользователя задается в БЕЙСИКе оператором:

DEF FN G(A,B,C,D,K,L,M,N) = USR address

В качестве адреса старта <sddress> здесь должен быть задан адрес, с которого в памяти размещается пусковая процедура START.

Вызов процедуры выполняется вызовом пользовательской функции, например так:

RANDOMIZE FN G(X1,X2,Y1,Y2,XH,YH,Zsc,FNaddr), где:

- X1, X2, Y1, Y2, XH, YH - см. на Рис. 34;
- Zsc - принятый масштаб для оси Z;
- FNaddr - адрес, в котором расположена Ваша процедура для расчета координаты Z - MYFUNC (см. ниже).

Буфер.

Мы организуем буфер, начиная с адреса BUFFER и отведем под него 256 байтов, т.е. по (BUFFER+255) включительно.

Вызываемые процедуры ПЗУ.

STACK_NUM (33B4H = 13236 DEC) - см. стр. 106.

FIND_INT2 (1E99H = 7833 DEC) - процедура снимает верхнее число со стека калькулятора и, полагая, что это целое число в

диапазоне 0...65535, передает его в регистровую пару BC. Ранее мы для этой цели всегда пользовались процедурой FP_TO_BC (2DA2H = 11682 DEC), что почти то же самое. Разница в том, что если при этом возникало переполнение, то FP_TO_BC соответственно выставляла флаг C регистра F, а процедура FIND_INT2 в этом случае вызывает процедуру REPORT_B (1E9FH=7839 DEC) для выдачи сообщения об ошибке "Integer out of range".

RST 30H - см. стр. 106.

BREAK_KEY (1F54H = 8020 DEC) - процедура предназначена для определение факта нажатия клавиши BREAK. Она выдает выключенный флаг C регистра F, если зафиксировано нажатие клавиши BREAK (или CAPS SHIFT + SPACE одновременно, что то же самое).

FP_TO_A (2DD5H = 11733 DEC) - см. стр. 106, 107.

PLOT (22DCH = 8924 DEC) - см. стр. 107.

Процедуры программы.

MYFUNC - эта процедура выполняет расчет по Вашей формуле z=f(....).

START - с этой процедуры начинается работа программы. Ее начало служит точкой входа. Здесь выполняется прием параметров из пользовательской функции FN() и размещение их на вершине стека калькулятора.

GRID - процедура является головной. Именно она увязывает в единое целое работу всех прочих процедур. Ее алгоритм имеет три логических части.

Первая часть выполняет инициализацию (первичную настройку). Так, например, Вы знаете, что "Спектрум" в исходном состоянии имеет только шесть ячеек памяти для своего калькулятора (M0...M5), но их число может быть увеличено до необходимых нам шестнадцати (M0...MF). Это выполняется именно здесь. Здесь же

заполняются семь ячеек памяти калькулятора (с M6 до MC). Данные для них берутся со стека калькулятора, куда попали из функции пользователя FN (). Здесь же рассчитывается масштабный коэффициент XYscale, необходимый для того, чтобы вычерчиваемая трехмерная поверхность не вышла за пределы экрана компьютера. Он заносится в ячейку памяти MD.

Вторая и третья части процедуры во многом похожи - это два прохода, при которых строится сама трехмерная поверхность. На первом проходе выстраиваются кривые вдоль оси Y, а на втором - вдоль оси X. Каждый из проходов фактически представляет из себя двойной вложенный цикл. Так, на первом проходе во внешнем цикле варьируется координата Y для каждой кривой, а во внутреннем цикле отрабатывается координата X. На втором (поперечном) проходе, соответственно, все происходит наоборот.

Обратите внимание на то, что при входе в эту подпрограмму, регистровая пара HL должна содержать адрес, начиная с которого размещается процедура для расчета формулы, по которой строится трехмерная поверхность. Этот адрес был сюда поставлен после работы процедуры START, которая, в свою очередь, приняла его из пользовательской функции FN ().

CLEAR - эта очень маленькая процедура служит для очистки нашего 256-байтного буфера. Очистка производится занулением всех его ячеек.

PLOT_3D - процедура служит для печати на экране начальных и конечных точек отрезков (узловых точек) трехмерной поверхности. Пространственные координаты X и Y она получает из ячеек памяти калькулятора ME и MF, а затем вызовом процедуры CONVERT преобразует их в двумерные экранные координаты.

DRAW_3D - процедура очень похожа на предыдущую. Если та процедура выпечатывала концы отрезков, из которых состоит поверхность, то эта процедура рисует сами отрезки. При этом пересчет трехмерных координат в плоские выполняется вызовом процедуры CONVERT, а печать точек, входящих в отрезки - вызовом процедуры DRAW_TO, которая и принимает решение о том, видима ли текущая точка (надо ли ее изображать).

Кроме того, процедура DRAW_3D выполняет еще одну полезную функцию. Она проверяет, не была ли нажата клавиша BREAK (CAPS SHIFT + SPACE). Это дает Вам возможность прервать построение трехмерной поверхности в случае необходимости.

CONVERT - очень важная и интересная процедура. Она конвертирует (преобразует) трехмерные координаты нашей поверхности в двумерные координаты экранной плоскости. Координаты X и Y трехмерной поверхности хранятся в ячейках памяти калькулятора ME и MF. Взяв их оттуда, процедура рассчитает с помощью подготовленной Вами процедуры третью координату Z, а затем пересчитает их в экранные координаты PlotX и PlotY (хранятся в M4 и M5) с помощью специального алгоритма. При выходе из процедуры экранные координаты PlotX и PlotY остаются на стеке калькулятора (в этом порядке).

Надо сказать несколько слов об алгоритме, с помощью которого трехмерные пространственные координаты пересчитываются в двухмерные экранные. На Рис. 29 показан трехмерный параллелепипед. Приведенное изображение является изометрической проекцией. Чтобы пересчитать по координатам (X, Y, Z) экранные координаты X, Y мы можем воспользоваться например следующими формулами:

$$X = SQR(3)*(Y-X)/2$$

$$Y = Z - (Y+X)/2$$

Расчет по этим формулам и заложен в работу процедуры CONVERT.

DRAW_TO - это разновидность подпрограммы, предназначенной для рисования отрезков прямых. От своих традиционных аналогов она отличается тем, что при печати каждой точки (из которых состоит отрезок) она обращается к нижеприведенной процедуре G_PLOT, которая решает надо ли данную точку печатать и печатает ее только после того, как сверится с буферной таблицей.

Для своей работы процедура DRAW_TO использует текущие экранные координаты начала отрезка, хранящиеся в ячейках памяти калькулятора M4 и M5. Конечные точки отрезка она снимает со стека калькулятора, ранее помещенные туда процедурой CONVERT где они хранятся в порядке Xend, Yend.

G_PLOT - эта процедура принимает решение о том, следует или нет печатать текущую точку и выполняет данную печать, если необходимо. Координаты позиции печати берутся из ячеек M4 и M5 памяти калькулятора. Решение о том, печатать или нет, принимается по результатам сравнения полученной вертикальной экранной координаты с соответствующим для данного столбца содержимым буфера. Принцип, по которому это делается, мы описали выше. Если точка не скрыта, то выполняется ее печать.

Процедура MYFUNC.

```
Код     Метка    Мнемоника        Комментарий (стек калькулятора)
EF      MYFUNC   RST 28           ;Включение калькулятора.
EE               recall ME        ;На вершине стека: X.
31               duplicate        ;X, X
04               multiply         ;X*X
EF               recall MF        ;X*X, Y
31               duplicate        ;X*X, Y,Y
04               multiply         ;X*X, Y*Y
0F               add              ;X*X + Y*Y
28               sqr              ;R, где R=SQR(X^2+Y^2)
31               duplicate        ;R, R
30               eq_zero          ;R, R=0?
0008          jump_true,ZERO      ;Если R=0, то этот случай надо
                                  ;обрабатывать особо; переходим
                                  ;на метку ZERO.
31               duplicate        ;R,R
1F               sin              ;R, SIN(R)
01               exchange         ;SIN(R), R
05               divide           ;SIN(R)/R
38               endcalc          ;Выключение калькулятора.
C9               RET              ;Выход.
EF      ZERO     delete           ;Стек пуст.
A1               const_one        ;1. Предел SIN(R)/R, когда
                                  ;R стремится к нулю, равен 1.
38               endcalc          ;Выключение калькулятора.
C9               RET              ;Выход.
```

Процедура START.

```
Код      Метка    Мнемоника     Комментарий (стек калькулятора)

2A0B5C   START    LD HL,(5CB0)  ;По адресу 5CB0H находится сис-
                                ;темная переменная DEFADD, ко-
                                ;торая в момент расчета функ-
                                ;ции пользователя содержит ад-
                                ;рес, с которого начинаются
                                ;параметры этой функции.
23       S_LOOP   INC HL        ;Пропустили имя параметра.
23                INC HL        ;Пропустили код CHR 14. Теперь
                                ;HL указывает на сам параметр,
                                ;записанный в пятибайтной фор-
                                ;ме.
CDB433            CALL 33B4     ;Вызов процедуры системного ПЗУ
                                ;STACK_NUM, см. с. 106.
7E                LD A,(HL)     ;Проверим, что за байт стоит
                                ;после принятого параметра.
23                INC HL        ;Пропустим этот байт.
2E2C              CP 2C         ;Не запятая ли это? Код запятой
                                ;равен 2CH.
28F5              JR Z,S_LOOP   ;Если это запятая, то значит не
                                ;все параметры еще приняты и мы
                                ;возвращаемся на метку S_LOOP.
CD991E            CALL 1E99     ;Вызов процедуры ПЗУ FIND_INT2.
                                ;Верхнее число со стека кальку-
                                ;лятора передается в пару BC, а
                                ;это число, как мы знаем, явля-
                                ;ется адресом для расчета нашей
                                ;функции Z=f(....).
60                LD H,B        ;Адрес нашей функции передается
69                LD L,C        ;в регистровую пару HL.
C3????            JP GRID       ;Безусловный переход на голов-
                                ;ную процедуру GRID для продол-
                                ;жения работы.
```

Процедура GRID.

```
Код       Метка   Мнемоника       Комментарии (стек калькулятора)

E5        GRID    PUSH HL         ;Запомнили на стеке адрес про-
                                  ;цедуры вычисления функции
                                  ;Z=f(...).
CD????            CALL CLEAR      ;Вызов процедуры CLEAR для
                                  ;очистки буферной таблицы.
015000            LD BC,0050H     ;Число 50H = 80 DEC - это необ-
                                  ;ходимое количество ячеек памя-
                                  ;ти для размещения 16-ти ячеек
                                  ;памяти калькулятора, поскольку
                                  ;в каждой ячейке должно хра-
                                  ;ниться пятибайтное число.
F7                RST 30H         ;Резервируется область в опера-
                                  ;тивной памяти на 80 ячеек.
EB                EX DE,HL        ;Теперь HL указывает на начало
                                  ;нулевой ячейки памяти кальку-
                                  ;лятора.
22685C            LD (MEM),HL     ;В системную переменную MEM
                                  ;(5C68H = 23656 DEC), указываю-
                                  ;шую на начало памяти калькуля-
                                  ;тора, заносится этот адрес.
0E41              LD C,41H        ;Число 41H = 65 DEC (13 ячеек
                                  ;памяти калькулятора.
09                ADD HL,BC       ;Теперь HL указывает на 14-ую
                                  ;ячейку калькулятора (MD).
EB                EX DE,HL        ;DE указывает на ячейку MD.
0E24              LD C,24H        ;24H = 36 DEC (7 пятибайтных
                                  ;ячеек + 1 байт).
2A655C          LD HL,(STKEND)    ;Системная переменная STKEND
                                  ;(5C65H=23653 указывает на
                                  ;ячеику, следующую за вершиной
                                  ;стека калькулятора).
EDB8              LDDR            ;Блочное перемещение. Перемеща-
                                  ;ются 36 байтов, т.е. 7 верхних
                                  ;вложений со стека калькулятора
```

```
                                ;копируются в нисходящем порядке в ячеики памяти калькулятора MC...M6.
23                      INC HL  ;При работе команды LDDR содержимое HL уменьшилось на 36, а
                                ;теперь увеличилось на 1. Т.о.
                                ;теперь HL указывает на начало
                                ;стека калькулятора.
22655C          LD (STKEND),HL  ;Совместив конец стека с его
                                ;началом, мы фактически очистили стек.
E1                      POP HL  ;Вернули в HL с машинного стека
                                ;адрес процедуры для расчета
                                ;искомой функции z=f(...) и за-
22925C          LD (MEMBOT),HL  помнили его в системной переменной MEMBOT (для использования в процедуре CONVERT - см. ниже).
```

Мы закончили первичную инициализацию ячеек памяти калькулятора и теперь рассчитаем масштабный фактор XYscale, для чего вначале надо будет заслать на стек калькулятора число:

256/SQR(3) ~ 149.7.

В пятибайтной форме оно может быть представлено, как:

88 13 CD 3A 2C.

```
Код             Метка   Мнемоника       Комментарий (стек калькулятора)

EF                      RST 28          ;Включение калькулятора.
34F813CD3A2C            stk_data        ;Поместили на вершину стека
                                        ;калькулятора число 256/SQR(3).
E8                      recall M8       ;На стеке: 256/SQR(3), X2
E6                      recall M6       ;256/SQR(3), X2, X1.
03                      subtract        ;256/SQR(3), X2-X1.
E9                      recall M9       ;256/SQR(3), X2-X1, Y2.
E7                      recall M7       ;256/SQR(3), X2-X1, Y2, Y1.
03                      subtract        ;256/SQR(3), X2-X1, Y2-Y1.
0F                      add             ;256/SQR(3), (X2-X1)+(Y2-Y1).
```

```
05                    divide       ;256/(SQR(3)*((X2-X1)+(Y2-Y1))).
CD                    store MD     ;Теперь в ячейке MD содержится
                                   ;масштабный фактор XYscale.
02                    delete       ;Стек пуст.
E9                    recall M9    ;Y2
CF                    store MF     ;Ячейка MF содержит Y=Y2.
02                    delete       ;Стек пуст.
E8          LOOP_Y1   recall M8    ;X2
CE                    store ME     ;Ячейка ME содержит X=X2.
02                    delete       ;Стек пуст.
38                    endcalc      ;Выключение калькулятора.
CD????                CALL PLOT_3D ;Вызов подпрограммы PLOT_3D
                                   ;для печати самой первой точки.
EF                    RST 28       ;Включение калькулятора.
EE          LOOP_X1   recall ME    ;На стеке: X.
EA                    recall MA    ;На стеке: X, XH.
03                    subtract     ;На стеке: X - XH.
CE                    store ME     ;Запомнили новое, уменьшенное
                                   ;значение X.
E6                    recall M6    ;На стеке: X, X1.
03                    subtract     ;На стеке: X - X1.
36                    lt_zero      ;Проверка X - X1 < 0?
0008            Jump_true EXIT_1   ;Переход, если цикл закончен.
38                    endcalc      ;Выключение калькулятора.
CD????                CALL DRAW_3D ;Вызов подпрограммы DRAW_3D для
                                   ;изображения очередного отрез-
                                   ;ка.
EF                    RST 28       ;Включеиие калькулятора.
33F1                  JUMP LOOP_X1 ;Переход в начало цикла.
EF          EXIT_1    recall MF    ;На стеке: Y (при работе в ко-
                                   ;дах калькулятора следует пом-
                                   ;нить, что переход Jump_true
                                   ;имеет косвенный эффект - с
                                   ;вершины стека снимается то,
                                   ;что там было, т.е. X-X1 на
                                   ;стеке больше нет.
EB                    recall MB    ;На стеке: Y, YH.
03                    subtract     ;На стеке: Y - YH.
```

```
CF                  store MF        ;Запомнили новое, уменьшенное
                                    ;значение Y.
E7                  recall M7       ;На стеке: Y, Y1.
03                  subtract        ;На стеке: Y - Y1.
36                  lt_zero         ;Проверка Y - Y1 < 0?
30                  not             ;Проверка Y - Y1 >= 0?
00DF          JUMP_true LOOP_Y1     ;Возврат, если цикл не закон-
                                    ;чен.
38                  endcalc         ;Выключение калькулятора.
CD????              CALL CLEAR      ;Очистка буферной таблицы.
EF                  RST 28          ;Включение калькулятора.
E8                  recall M8       ;На стеке: X2.
CE                  store ME        ;Ячейка ME содержит X=X2.
02                  delete          ;Стек пуст.
E9          LOOP_X2 recall M9       ;Y2
CF                  store MF        ;Ячейка MF содержит Y=Y2.
02                  delete          ;Стек пуст.
38                  endcalc         ;Выключение калькулятора.
CD????              CALL PLOT_3D    ;Вызов подпрограммы PLOT_3D
                                    ;для печати первой точки на
                                    ;втором проходе.
EF                  RST 28          ;Включение калькулятора.
EF          LOOP_Y2 recall MF       ;На стеке: Y.
EB                  recall MB       ;На стеке: Y, YH.
03                  subtract        ;На стеке: Y - YH.
CF                  store MF        ;Запомнили новое, уменьшенное
                                    ;значение Y.
E7                  recall M7       ;На стеке: Y, Y1.
03                  subtract        ;На стеке: Y - Y1.
36                  lt_zero         ;Проверка Y - Y1 < 0?
0008          JUMP_true EXIT_2      ;Переход, если цикл закончен.
38                  endcalc         ;Выключение калькулятора.
CD????              CALL DRAW_3D    ;Вызов подпрограммы DRAW_3D для
                                    ;изображения очередного отрез-
                                    ;ка.
EF                  RST 28          ;Включение калькулятора.
33F1                JUMP LOOP_Y2    ;Переход в начало цикла.
EE          EXIT_2  recall ME       ;X
```

```
EA                recall MA      ;X, XH.
03                subtract       ;X-XH.
CE                store ME       ;Запомнили новое, уменьшенное
                                 ;значение X.
E6                recall M6      ;X, X1.
03                subtract       ;X-X1.
36                lt_zero        ;X<X1?
30                not            ;X>=X1?
OODF          JUMP_true LOOP_X2  ;Возврат, если цикл не
                                 ;закончен.
38                endcalc        ;Выключение калькулятора.
21925C            LD HL,MEMBOT
22685C            LD (MEM),HL    ;Восстановление адреса ячеек
                                 ;памяти калькулятора.
C9                RET            ;Возврат
```

Процедура CLEAR.

Код	Метка	Мнемоника	Комментарии (стек калькулятора)
21????	CLEAR	LD HL,BUFFER	;Адрес начала буфера.
11????		LD DE,BUFFER+1	;
01FF00		LD BC,FFH	;FFH = 255 DEC. Подготовка к ;обнулению 255 байтов.
70		LD (HL),B	;Обнулили первый байт буфера.
EDB0		LDIR	;Обнулили остальные 255 байтов.
C9		RET	;Выход из процедуры.

Процедура PLOT_3D.

Код	Метка	Мнемоника	Комментарии (стек калькулятора)
CD????	PLOT_3D	CALL CONVERT	;Вызов процедуры CONVERT для ;расчета экранных координат. ;На стеке: plotX, plotY.

```
EF              RST 28          ;Включение калькулятора.
C5              store M5        ;Переброска plotY в M5.
02              delete          ;На стеке: plotX.
C4              store M4        ;Переброска plotX в M4.
02              delete          ;Стек очищен.
38              endcalc         ;Выключение калькулятора.
C9              RET             ;Возврат в вызывающую подпро-
                                ;грамму.
```

Процедура DRAW_3D.

Код	Метка	Мнемоника	Комментарий (стек калькулятора)
CD????	DRAW_3D	CALL CONVERT	;Вызов подпрограммы CONVERT для ;расчета экранных координат.
CD????		CALL DRAW_TO	;Вызов подпрограммы DRAW_TO для ;рисования отрезка.
CD54F1		CALL 1F54H	;По адресу 1F54H (8020 DEC) в ;ПЗУ находится процедура ;BREAK_KEY. Вызов служит для ;проверки нажатия BREAK.
D27B1B		JP NC,1B7BH	;В этом случае выполняется пе- ;реход в ПЗУ на процедуру ;REPORT_L (1B7BH = 7035 DEC), ;которая запустит процедуру ;обработки ошибок RST 08 с ;кодом перехвата 14H, что озна- ;чает "BREAK into program".
C9		RET	;Возврат

Процедура CONVERT.

Код	Метка	Мнемоника	Комментарий (стек калькулятора)
2A925C	CONVERT	LD HL,(MEMBOT)	В регистровую пару HL засы- ;лается адрес, с которого начи- ;нается процедура для вычисле-

```
                                ;ния координаты Z. Этот адрес
                                ;был установлен в системной пе-
                                ;ременной MEMBOT (23698 DEC =
                                ;5C92H) с помощью процедуры
                                ;GRID, см. выше.
CD2C16          CALL 162C       ;В ПЗУ по этому адресу записана
                                ;одна инструкция JP (HL). Таким
                                ;образом, вызов адреса 162C эк-
                                ;вивалентен вызову той процеду-
                                ;ры, адрес которой установлен в
                                ;HL. Этот прием позволяет сде-
                                ;лать "нечувствительной" проце-
                                ;дуру CONVERT к тому, где нахо-
                                ;дится расчетная процедура.
EF              RST 28          ;Включение калькулятора. На
                                ;вершине стека координата Z
                                ;(установлена пользовательской
                                ;процедурой).
EC              recall MC       ;На стеке: Z, Zscale.
04              multiply        ;На стеке: Z', где Z'=Z*Zscale.
E8              recall M8       ;На стеке: Z', X2.
E9              recall M9       ;На стеке: Z', X2, Y2.
0F              add             ;На стеке: Z', X2+Y2.
EE              recall ME       ;На стеке: Z', X2+Y2, X.
EF              recall MF       ;На стеке: Z', X2+Y2, X, Y.
0F              add             ;На стеке: Z', X2+Y2, X+Y.
03              subtruct        ;На стеке: Z', (X2+Y2)-(X+Y)
ED              recall MD       ;Z', (X2+Y2)-(X+Y), XYscale
04              myltiply        ;Z', ((X2+Y2)-(X+Y))*XYscale
0F              add             ;Z'+ ((X2+Y2)-(X+Y))*XYscale
A2              const_half      ;Z'+ ((X2+Y2)-(X+Y))*XYscale,1/2
04              multiply        ;Z'+ ((X2+Y2)-(X+Y))*XYscale/2
27              int             ;ploty
```

Как видите, в результате операции получено значение, соответствующее экранной координате Y (см. с. 133 - формула (2)). Естественная разница состоит только в том, что X, Y и Z были нами домножены на масштабные коэффициенты Zscale и XYscale, но

это сделать было необходимо, чтобы трехмерная поверхность не вышла за пределы экрана. Таким образом, в результате последней операции выделения целой части мы фактически получили на вершине стека экранную координату PlotY. Теперь разберемся с координатой PlotX.

```
E8              recall M8     ;PlotY, X2
E7              recall M7     ;PlotY, X2, Y1
03              subtruct      ;PlotY, X2 - Y1
EE              recall ME     ;PlotY, X2 - Y1, X
EF              recall MF     ;PlotY, X2 - Y1, X, Y
03              subtruct      ;PlotY, X2 - Y1, X - Y
03              subtruct      ;PlotY, (X2 - Y1)-(X - Y)
34F15DB3D743    stk_data      ;PlotY, (X2-Y1)-(X-Y), SQR(3)
                              ;Эта операция помещает на вер-
                              ;шину стека число, равное корню
                              ;квадратному из трех. В пяти-
                              ;байтной форме оно имеет вид:
                              ;81 5D B3 D7 43.
04              multiply      ;PlotY, ((X2-Y1)-(X-Y))*SQR(3)
ED              recall MD     ;PlotY,((X2-Y1)-(X-Y))*SQR(3),
                              ;XYscale.
04              multiply      ;PlotY,((X2-Y1)-(X-Y))*SQR(3)*
                              ;XYscale.
A2              const_half    ;PlotY,((X2-Y1)-(X-Y))*SQR(3)*
                              ;XYscale, 1/2.
04              multiply      ;PlotY,((X2-Y1)-(X-Y))*SQR(3)*
                              ;XYscale/2.
27              int           ;PlotY, PlotX.
```

Сравнив полученное выражение с формулой (2), с. 133, Вы увидите, что на вершине стека практически получена экранная координата X = PlotX.

```
01              exchange      PlotX, PlotY
38              endcalc       Выключение калькулятора.
C9              RET           Возврат в вызывающую процедуру.
```

Процедура DRAW_TO.

```
Код     Метка     Мнемоника        Комментарий (стек калькулятора)

EF      DRAW_TO   RST 28           ;Включение калькулятора. На
                                   ;стеке содержатся (Xend, Yend).
E5                recall M5        ;(Xend, Yend, plotY).
C3                store M3         ;Запомнили в M3 plotY. На сте-
                                   ;ке:  (Xend, Yend, plotY).
03                subtract         ;Вычитание PlotY - Yend =
                                   ;=drawY.  (Xend, drawY).
E1                recall M1        ;Запомнили в M1 drawY. На сте-
                                   ;ке:  (Xend, drawY).
01                exchange         ;На стеке: (drawY, Xend).
E4                recall M4        ;(drawY, Xend, plotX)
C2                store M2         ;Запомнили в M2 plotX. На сте-
                                   ;ке:  (drawY, Xend, plotX)
03                subtract         ;Вычитание PlotX - Xend =
                                   ;=drawX.  (drawY, drawX).
C0                store M0         ;Запомнили в M0 drawX.
                                   ;(drawY, drawX).
07                or               ;Напомним, что нулевые значения
                                   ;на вершине стека калькулятора
                                   ;могут трактоваться как логи-
                                   ;ческое значение "ЛОЖЬ", а не-
                                   ;нулевые - как "ИСТИНА".  После
                                   ;операции "or" на вершине стека
                                   ;будет 0 только если оба значе-
                                   ;ния были равны нулю одновре-
                                   ;менно. Если же хоть одно из
                                   ;них не ноль, то результат
                                   ;будет "1" ("ИСТИНА").
0004          jump true CONT_1     ;Переход, если хоть одно из
                                   ;смещений не равно нулю. Значе-
                                   ;ние с вершины стека снимается.
38                endcalc          ;В противном случае выключение
                                   ;калькулятора.
C9                RET              ;Выход из программы.
```

```
E1          CONT_1   recall M1      ;Вызов на стек drawY.
2A                   abs            ;Вычисление абсолютной
                                    ;величины ABS (drawY).
E0                   recall M0      ;Вызов drawX. На стеке:
                                    ;ABS(drawY), drawX.
2A                   abs            ;ABS(drawY), ABS(drawX)
03                   subtruct       ;ABS(drawY)-ABS(drawX) - вычис-
                                    ;ление разности.
36                   lt_zero        ;Сравнение разности с нулем,
                                    ;т.е. как бы проверка что
                                    ;больше ABS(drawY) или
                                    ;ABS(drawX). В результате на
                                    ;стеке логическии 0 или 1. На-
                                    ;зовем это логическое значение,
                                    ;например LV (logical value).
31                   duplicate      ;Копирование вершины стека. В
                                    ;результате на стеке: LV,LV.
000D            jump true CONT_2    ;Переход, если LV = 1, т.е. ес-
                                    ;ли ABS(drawX) > ABS(drawY) и
                                    ;наш отрезок имеет горизонталь-
                                    ;ный характер.
E0                   recall M0      ;Вызов drawX. На стеке:
                                    ;LV, drawX.
E1                   recall M1      ;На стеке: LV, drawX, drawY.
C0                   store  M0      ;Перебросили drawY в M0
02                   delete         ;и удалили это значение со
                                    ;стека. На стеке: LV, drawX.
C1                   store  M1      ;Перебросили drawX в M1
02                   delete         ;и удалили это значение со
                                    ;стека.  На стеке: LV.
```

Смысл приведенных операций по переброске drawX из M0 в M1, а drawY из M1 в M0, если отрезок имеет вертикальный характер, состоит в том, чтобы установить в M0 то смещение из двух, которое имеет максимальное значение и тогда можно предсказать заранее сколько точек будет в отрезке. Число этих точек равно большему из смещений и фактически в любом случае может быть взято из M0.

Точно так же мы выполняем и переброску данных между M2 и M3.

```
E2              recall M2     ;На стеке: LV, lineX
E3              recall M3     ;На стеке: LV, lineX, lineY
C2              store  M2     ;На стеке: LV, lineX, lineY
02              delete        ;На стеке: LV, lineX
C3              store  M3     ;На стеке: LV, lineX
02              delete        ;На стеке: LV
```

После всего этого мы можем считать, что в ячейках M0 и M1 у нас содержатся смещения dispF и dispT, а в ячейках M2 и M3 - координаты lineF и lineT (см. Примечание к таблице использованных ячеек калькулятора - с. 149).

```
E1        CONT_2  recall M1     ;На стеке: LV, dispT
E0                recall M0     ;На стеке: LV, dispT, dispF
05                divide        ;Деление. На стеке:
                                ;LV, dispT/dispF
C1                store  M1     ;В ячейке M1 выставили переме-
                                ;ную ratio (см. табл. на с. 148).
02                delete        ;На стеке: LV.
E0                recall M0     ;На стеке: LV, dispF.
38                endcalc       ;Выключение калькулятора и
                                ;переход в коды Z-80.
00                NOP           ;Пауза.
CDD52D            CALL 2DD5     ;Вызов программы ПЗУ FP_TO_A.
                                ;Она снимает значение dispF с
                                ;вершины стека калькулятора и
                                ;помещает его в регистр A.
                                ;Ее значение равно количеству
                                ;точек, которые следует напе-
                                ;чатать.
F5        LOOP    PUSH AF       ;Запомнили на машинном стеке.
2009              JR NZ, NEG    ;В результате работы процедуры
                                ;FP_TO_A выключенный флаг Z ре-
                                ;гистра F говорит о том, что
                                ;число, снятое со стека в акку-
```

```
                                    ;мулятор - отрицательно, т.е.
                                    ;данный отрезок кривой рисуется
                                    ;задом наперед. В этом случае
                                    ;выполняется переход на метку
                                    ;NEG.
EF              RST 28              ;Включили калькулятор.
E2              recall M2           ;На стеке: LV, lineF
A1              const_one           ;На стеке: LV, lineF, 1.
0F              add                 ;На стеке: LV, lineF + 1.
E3              recall M3           ;На стеке: LV, lineF+1, lineT.
E1              recall M1           ;LV, lineF+1, lineT, ratio.
0F              add                 ;LV, lineF+1, lineT + ratio.
3308            Jump CONT_3         ;Обход на метку CONT_3.
EF      NEG     RST 28              ;Включили калькулятор.
E2              recall M2           ;На стеке: LV, lineF
A1              const_one           ;На стеке: LV, lineF, 1.
03              subtruct            ;На стеке: LV, lineF - 1.
E3              recall M3           ;На стеке: LV, lineF-1, lineT.
E1              recall M1           ;LV, lineF-1, lineT, ratio.
03              subtruct            ;LV, lineF-1, lineT - ratio.
C3      CONT_3  store M3            ;M3 и M5 получают координату
C5              store M5            ;печати точки в "поперечном"
                                    ;направлении.
02              delete              ;На стеке: LV, lineF+1
                                    ;или       LV, lineF-1.
C2              store M2            ;M2 и M4 получают координату
C4              store M4            ;печати точки в "прямом"
                                    ;направлении.
02              delete              ;На стеке: LV.
31              duplicate           ;На стеке: LV, LV.
0007        Jump true CONT_4        ;Переход вперед, если отрезок
                                    ;имеет горизонтальный характер.
                                    ;В противном случае меняем мес-
                                    ;тами содержимое M4 и M5.
E4              recall M4           ;На стеке: LV, plotF.
E5              recall M5           ;На стеке: LV, plotF, plotT.
C4              store  M4           ;Перебросили plotT в M4.
02              delete              ;На стеке: LV, plotF.
```

```
C5                  store  M5      ;Перебросили plotF в M5.
02                  delete         ;На стеке: LV.
38          CONT_4  endcalc        ;Выключение калькулятора.
CD????              CALL G_PLOT    ;Вызов процедуры проверки и пе-
                                   ;чати точки, координаты которои
                                   ;находятся в ячеиках M4 и M5.
C1                  POP BC         ;Ранее мы на стек отправляли
                                   ;AF, где регистр A содержал
                                   ;количество точек, входяших в
                                   ;отрезок. Теперь это число по-
                                   ;мешается в регистр B.
05                  DEC B          ;Уменьшили счетчик на 1
C5                  PUSH BC        ;и запомнили его на стеке.
2803                JR Z,EXIT      ;Если счетчик исчерпан, то пе-
                                   ;реход на EXIT для подготовки
                                   ;к выходу.
F1                  POP AF         ;В противном случае рисование
                                   ;отрезка надо продолжать. Необ-
                                   ;ходимость восстанавливать AF
                                   ;вызвана тем, что в регистре F
                                   ;флаг Z содержит информацию о
                                   ;направлении отрезка.
18D2                JR LOOP        ;Возврат в начало цикла для
                                   ;печати последуюших точек
                                   ;отрезка.
EF          EXIT    RST 28         ;Включили калькулятор.
02                  delete         ;Перед выходом очистили его
                                   ;стек.
38                  endcalc        ;Выключили калькулятор.
F1                  POP AF         ;Очистили машинныи стек.
C9                  RET            ;Возврат.
```

Процедура G_PLOT

```
Код         Метка   Мнемоника     Комментарии (стек калькулятора)

EF          G_PLOT  RST 28         ;Включение калькулятора.
E4                  recall M4      ;(plotX)
```

```
E5                  recall M5     ;(PlotX,PlotY)
38                  endcalc       ;Выключение калькулятора.
CDD52D              CALL 2DD5     ;Вызов программы ПЗУ FP_TO_A.
                                  ;Она снимает значение с вершины
                                  ;стека калькулятора и помещает
                                  ;его в регистр А процессора.
                                  ;Теперь там переменная PlotY.
F5                  PUSH AF       ;Запомнили ее на машинном сте-
                                  ;ке.
CDD52D              CALL 2DD5     ;Вызов программы ПЗУ FP_TO_A.
                                  ;В регистре А - PlotX.
3819                JR C,EXIT     ;Переход, если координата X
                                  ;слишком велика.
2017                JR NZ,EXIT    ;Переход, если X - меньше ну-
                                  ;ля.
6F                  LD L,A        ;В регистре L - координата X.
26??                LD H,??H      ;В регистр H заносим старшии
                                  ;байт буферной таблицы.
F1                  POP AF        ;В аккумуляторе теперь PlotY.
C0                  RET NZ        ;Выход, если координата Y
                                  ;отрицательна.
3804                JR C,MAXIM    ;Переход, если координата Y
                                  ;слишком велика (>255).
FEB0                CP B0H        ;B0H = 176 (предел по верти-
                                  ;кали)
3802                JR C,PL_OK    ;Переход, если Y<176 и все в
                                  ;порядке.
3EFF      MAXIM     LD A,FF       ;В противном случае в аккумуля-
                                  ;торе выставляем "FF", как сиг-
                                  ;нал о выходе за пределы экра-
                                  ;на.
BE        PL_OK     CP(HL)        ;Сравним с тем, что содержится
                                  ;в буфере.
D8                  RET C         ;Выход, если точка должна быть
                                  ;"невидимой" и ее печатать не
                                  ;надо.
77                  LD (HL),A     ;В противном случае запомнили в
                                  ;буфере новое значение для
```

```
                                ;данной вертикали.
47              LD B,A          ;
4D              LD C,L          ;Теперь BC содержит экранные
                                ;координаты печатаемой точки.
3C              INC A           ;Проверка на выход за пределы
                                ;экрана.
C4E522          CALL NZ,22E5    ;Вызов подпрограммы ПЗУ PLOT.
                                ;которая напечатает точку, ес-
                                ;ли она не лежит вне пределов
                                ;экрана.
C9              RET             Возврат в вызывающую программу.
```

4. БЛОЧНАЯ ГРАФИКА

Выше мы упомянули, что воспроизведение блочной графики сводится на практике к печати символов, размещенных либо в ПЗУ, либо в ОЗУ компьютера. Это наиболее простой и экономичный путь получения сложных графических изображений. Разумеется в большинстве случаев в качестве печатаемых символов выступают не символы стандартного набора ASCII, а заранее подготовленные программистом специальные символы, на расположение которых в ОЗУ указывают с помощью системной переменной CHARS (23606 DEC = 5C36H).

Приняв в своей программе именно такой метод воспроизведения графики, Вы значительно облегчаете себе жизнь, благодаря тому, что можете использовать стандартные процедуры ПЗУ для печати символов на экране. Это прежде всего процедура, которая задействуется командой процессора RST 10H и некоторые другие процедуры, опирающиеся на нее. В основном упрощение программирования достигается за счет того, что Вам теперь уже не нужно для печати объекта на экране производить пересчеты экранных координат в адреса экранной памяти, т. к. при использовании RST 10H Вы можете задавать эти координаты напрямую.

В предыдущем томе - "Элементарная графика" мы достаточно подробно остановились на печати с помощью RST 10H и теперь имеем возможность на этом вопросе не останавливаться, а рассмотреть некоторые дополнительные приемы. Тем не менее, в двух словах мы все же напомним основы печати через RST 10H.

1. Для того, чтобы напечатать с помощью команды RST 10H некоторый символ, необходимо:

- установить код этого символа в аккумуляторе, что делается тривиально просто:

```
LD A,N
```

- сделать текущим канал, по которому должна выполняться печать. Это можно сделать несколькими способами.

Во-первых, Вы можете применить для этого системную процедуру ПЗУ CHAN_OPEN (1601H = 5633 DEC). Перед тем, как ее вызывать, в аккумуляторе следует установить номер потока, которыи подключен к этому каналу. Так, для печати в основную часть экрана (канал "S", поток #2) там следует установить число 2. А для печати в нижнюю часть экрана (канал "K", поток #0) там следует установить 0.

Во-вторых, Вы можете воспользоваться для этой цели системной переменной TV_FLAG (5C3CH = 23612 DEC), нулевой бит которой указывает на то, какой канал вывода в данныи момент является текущим. Если этот бит выключен, то текущим будет канал "S", а если включен, то канал "K".

2. Применяя управляющие коды, Вы можете управлять как позициеи печати, так и цветовыми атрибутами печатаемых символов. Управляющие коды (управляющие символы) выдаются командои RST 10H точно так же, как и печатные символы.

3. Если Вы хотите напечатать не один символ, а последовательность символов (символьную строку), то можете воспользоваться процедурои ПЗУ PR_STRING (203CH = 8252 DEC). Печатаемая символьная строка может включать в себя и коды управления цветом и позициеи печати. Т.к. PR_STRING в своей работе опирается на RST 10H, то соответственно здесь тоже необходимо предварительно задавать текущий канал вывода.

4.1. Создание аркадных эффектов в блочной графике.

Поскольку экранное изображение в программах строится из заранее заданных пользователем символьных блоков 8 X 8, здесь есть элегантная возможность введения в игру аркадных атрибутов. Выше мы говорили о том, что в программах делают это "подвешивая" аркадные свойства объектов на их цветовые атрибуты или, скажем, вводя "теневой" экран аркадных атрибутов. В программах, использующих символьную графику (правильнее ее считать псевдографикой) нередко аркадные атрибуты "подвешивают" на пиксельную конструкцию псевдосимволов. Несколько примеров такого подхода

мы сейчас и рассмотрим. Вы можете и сами посмотреть, как это организовано, например в программах MANIC MINER, JET SET WILLI.

В основу этой технологии положен тот факт, что объекты, имеющие разные аркадные свойства могут строиться из различных блоков. Рассмотрим некоторые виды специальных блоков.

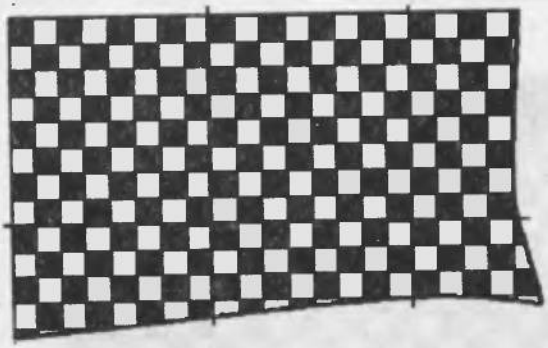


Рис. 35

Это основной блок, из которого строится изображение игрового поля. При вертикальном перемещении героя такой блок является "опорой" - на нем можно стоять, не проваливаясь. При горизонтальном перемещении этот блок непроходим. Идентифицировать наличие такого блока в знакоместе с координатами X,Y - очень просто. Надо по координатам знакоместа найти адрес, соответствующий его верхней пиксельной линии и, если по этому адресу содержится число 170, значит это он.

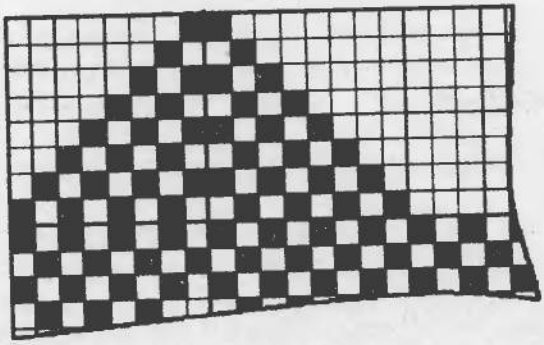


Рис. 36

На Рис. 36 показаны примеры блоков, являющихся опорой другого рода - они выполняют функции наклонных плоскостей. Если под героем идентифицирован блок, верхняя линия которого равна 1 или 128, включается иная процедура, обрабатывающая движение. Так, при перемещении вправо/влево, одновременно должна изменяться и вертикальная координата героя. В программе JET SET WILLI эта процедура устроена таким образом, что когда герой "прыгает" эта опора является проницаемой.

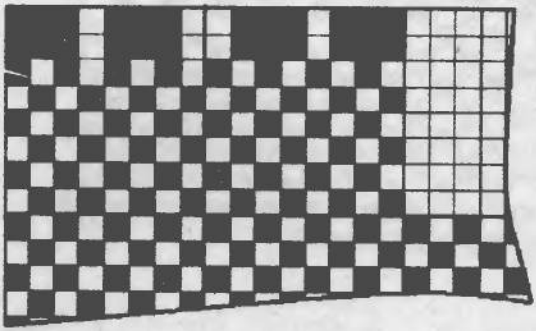


Рис. 37

На рис. 37 показаны два примера бегущих дорожек. Такие блоки "включают" процедуры самопроизвольного движения героя вправо или влево. Идентифицируются они по наличию числа 238 или 119 в адресе, соответствующем первой линии знакоместа.

Если эта логика понятна, то нашему читателю не составит труда самому подготовить конструкцию "смертельных" блоков - "огонь", "вода", "сосульки", "колючки" и т. д. и т. п.

Основной цикл работы программы выглядит следующим образом:

- определяем, какие управляющие клавиши нажаты;
- проверяем возможность перемещения в заданном направлении;
- проверяем, что за блок находится под новой координатой героя (направление "вниз" надо проверять всегда, даже

если клавиша "вниз" не нажималась - это происходит благодаря закону всемирного тяготения);
- если никаких особенностей не выявлено, герои печатается в новой координате;
- если выявлен специфический блок, запускаются специальные процедуры, которые могут, например, включить подпрограмму обработки гибели героя.

4.2. Создание трехмерных эффектов в блочной графике.

Для программ, работающих с псевдографикой, наиболее простым способом создания эффекта трехмерного пространства является наложение теней, причем делается это программно с помощью специальной подпрограммы-генератора тени. Можно, конечно, предусмотреть тени и в самой картинке, но это не столь удобно, гораздо проще поручить эту заботу генератору.

Рассмотрим создание такого генератора на конкретном примере. Его задача - просканировать экран и, если на экране будут обнаружены "твердые" объекты, выполнить "оконтуривание" этих объектов тенью. Будем считать, что освещение экрана направлено слева-сверху. В этом случае все твердые тела должны отбрасывать тень вправо-вниз.

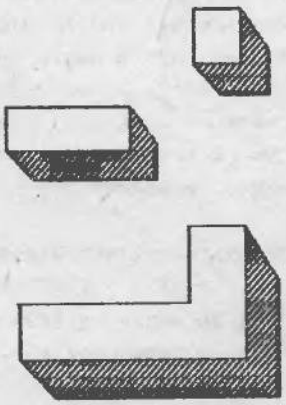


Рис. 38

Введем понятие "твердого тела". Таким телом будем считать всякое тело, имеющее пиксельный шаблон такой, какой показан на Рис. 35. Для идентификации того, что контур рисовать надо, мы можем использовать верхнюю пиксельную линию, которая у такого тела должна равняться 170.

Тень выглядит изящнее, если ее ширина не равна символьному блоку (8 пикселов), а несколько меньше - 3...5 пикселов. Можно сказать так, что "широкие" тени больше соответствуют дизайну ранних программ, а "узкие" - дизайну программ последних лет выпуска. Мы исполним тени шириной 4 пиксела. Для такого генератора нам потребуются три "теневых элемента". По одному для горизонтальных и вертикальных теней и один элемент в качестве углового, см. Рис. 39.

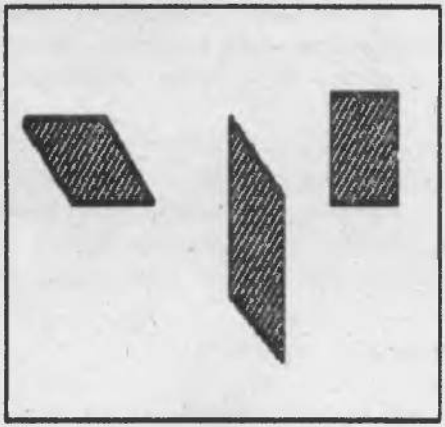


Рис. 39

Список процедур.

SHADOW - головная процедура. Здесь выполняется сканирование экрана. Поскольку тень должна отбрасываться вправо-вниз, нам удобнее работать, если мы начием сканировать экран и строить теневой контур из правого нижнего угла, следуя по порядку справа-налево и снизу-вверх. В случае, если находится знакоместо, в котором байт верхней линии равен 170 (0AAH), то после необходимых проверок, связанных с границами экрана, вызываются

процедуры VERT или HORIZ для рисования вертикальной или горизонтальной тени.

ADDR - процедура рассчитывает адрес в экранной области для верхней линии знакоместа, координаты которого (в знакоместах) выставлены в регистровой паре DE. Результат возвращается в регистровой паре HL.

VERT - процедура, оттеняющая "твердый" блок справа по вертикали. Мы приходим сюда с адресом знакоместа, в котором должна быть напечатана "тень", установленным в регистровой паре HL. Печать осуществляется наложением по OR байта, в котором включены четыре старших бита. Это выполняется в цикле из 8 шагов. Процедура несколько усложнена тем, что надо проверить, не является ли исследуемый блок угловым. Если это так, то тогда надо напечатать еще и "угловую тень", которая печатается в цикле из четырех шагов.

HORIZ - процедура, оттеняющая блок снизу. Печать осуществляется включением всех битов в четырех верхних линиях знакоместа.

Процедура SHADOW.

```
Код       Метка     Мнемоника     Комментарий

F3        SHADOW    DI            ;Отключение прерываний.
0616                LD B,16H      ;Организация цикла для 22
C5        LOOP      PUSH BC       ;строк.
58                  LD E,B        ;
1D                  DEC E         ;Текущая координата Y.
0620                LD B,20H      ;Организация цикла для 32
C5        LOOP_1    PUSH BC       ;столбцов.
50                  LD D,B        ;
15                  DEC D         ;Текущая координата X.
CD????              CALL ADDR     ;Вызов процедуры для расчета
                                  ;адреса в дисплеином файле для
                                  ;знакоместа, заданиого в DE.
7E                  LD A,(HL)     ;Байт в первой линии текущего
```

```
                                    ;знакоместа.
FEAA                CP AAH          ;Он равен 170?
2026                JR NZ,PASS      ;Если нет, то переход на PASS.
7A                  LD A,D
3C                  INC A           ;Проверим знакоместо справа.
FE1F                CP 20H          ;
3020                JR NC,PASS      ;Если это самый правый столбец
                                    ;экрана, то оттенять его не
                                    ;надо.
2C                  INC L           ;Переход к знакоместу справа.
7E                  LD A,(HL)       ;Что там содержится?
FEAA                CP AAH          ;Байт равен 170?
2803                JR Z,PASS_1     ;Если да, то это не то, что нам
                                    ;нужно, переходим на PASS_1.
CD????              CALL VERT       ;Вызов процедуры вертикального
                                    ;оконтуривания.
2D          PASS_1  DEC L           ;Восстановили L
7B                  LD A,E          ;
3C                  INC A           ;Проверим знакоместо снизу.
FE16                CP 16H          ;
3010                JR NC,PASS      ;Если это нижняя строка из 22
                                    ;возможных, то ее оттенять не
                                    ;надо.
7D                  LD A,L          ;Проверим, что содержится
C620                ADD A,20H       ;в знакоместе снизу.
67                  LD L,A          ;
7E                  LD A,(HL)       ;
FEAA                CP AAH          ;Байт равен 170?
2803                JR Z,PASS_2     ;Если да, то это не то, что
                                    ;нам нужно, переходим к PASS_2.
CD????              CALL HORIZ      ;Вызов процедуры оттенения
                                    ;по горизонтали.
7D          PASS_2  LD A,L          ;Восстановили
D620                SUB 20H         ;предыдущее
67                  LD L,A          ;значение L.
C1          PASS    POP BC          ;Восстановление счетчика
                                    ;столбцов.
10CC                DJNZ LOOP_1     ;Переход к столбцу слева.
```

```
C1              POP BC          ;Восстановление счетчика рядов.
10C4            DJNZ LOOP       ;Переход к верхнему ряду.
FB              EI              ;Включение прерываний.
C9              RET             ;Выход из подпрограммы.
```

Процедура ADDR.

```
Код     Метка   Мнемоника       Комментарий

7B      ADDR    LD A,E          ;Координата Y.
E618            AND 18H         ;Определяем номер сегмента эк-
                                ;рана.
F640            OR 40H          ;Выставляем начальный адрес
                                ;дисплейного файла.
67              LD H,A          ;Результат выставили в H.
7B              LD A,E          ;Координата Y.
E607            AND 07          ;Номер ряда в сегменте.
B7              OR A            ;Сброс флага переноса.
1F              RRA             ;Перемещаем номер
1F              RRA             ;ряда в три старших
1F              RRA             ;бита.
1F              RRA             ;
82              ADD A,D         ;Заполняем пять младших
                                ;битов номером столбца.
6F              LD L,A          ;Результат выставляем в L.
C9              RET             ;Возврат.
```

Процедура VERT.

```
Код     Метка   Мнемоника       Комментарий

E5      VERT    PUSH HL         ;Запомнили HL на стеке.
0608            LD B,08         ;Счетчик на 8 байтов для верти-
                                ;кального блока.
7E      LOOP    LD A,(HL)       ;Взяли то, что уже есть на
                                ;экране в текущем знакоместе.
```

```
F6F0            OR F0H          ;Наложили на него вертикальный
                                ;блок.
77              LD (HL),A       ;Поместили результат в текущее
                                ;знакоместо.
24              INC H           ;Перешли к новой линии.
10F9            DJNZ LOOP       ;Если не все 8 линий перераба-
                                ;таны, то возврат на LOOP.

7B              LD A,E          ;Номер текущей строки.
3C              INC A           ;Проверка, не
6F              LD L,A          ;последняя ли
7E              LD A,(HL)       ;это строка.
FE16            CP 16H          ;
3014            JR NC,PASS      ;Если да, то обход.
E1              POP HL          ;Восстановили со стека адрес
                                ;первой линии знакоместа.
E5              PUSH HL         ;И снова его запомнили.
7D              LD A,L          ;Перешли к первой линии в
C620            ADD A,20H       ;нижележащем
6F              LD L,A          ;знакоместе.
7E              LD A,(HL)       ;Что там содержится?
FEAA            CP AAH          ;Если там число 170, то здесь
2809            JR Z,PASS       ;печатать угловую тень не на-
                                ;до. Во всех же остальных
                                ;случаях она здесь нужна.
0604            LD B,04         ;Цикл в 4 шага для печати уг-
                                ;лового блока.
7E      LOOP_1  LD A,(HL)       ;Взяли то, что есть в текущей
                                ;линии.
F6F0            OR F0H          ;Наложили угловой блок.
77              LD (HL),A       ;Поместили на экран.
24              INC H           ;Перешли к новой линии.
10F9            DJNZ LOOP_1     ;Если не все линии напечатаны,
                                ;возврат на LOOP_1.
E1      PASS    POP HL          ;Перед выходом восстановили
                                ;исходное содержимое HL.
C9              RET             ;Возврат.
```

Процедура HORIZ.

```
Код     Метка   Мнемоника       Комментарий

E5      HORIZ   PUSH HL         ;Запомнили HL на стеке.
0604            LD B,04         ;Счетчик на 4 байта для гори-
                                ;зонтального блока.
7E      LOOP    LD A,(HL)       ;Взяли то, что уже есть на
                                ;экране в текущем знакоместе.
F6FF            OR FFH          ;Наложили на него горизонталь-
                                ;ный блок.
77              LD (HL),A       ;Поместили результат в текущее
                                ;знакоместо.
24              INC H           ;Перешли к новой линии.
10F9            DJNZ LOOP       ;Если не все 4 линии перераба-
                                ;таны, то возврат на LOOP.
E1              POP HL          ;Восстановление HL перед выхо-
                                ;дом из процедуры.
C9              RET             ;Возврат.
```

ПРИЛОЖЕНИЕ 1

Генератор нестандартных символов.

Случаи, когда Вам могут потребоваться собственные шрифты - очень нередки. Во-первых, самый типичный случай, когда Вас не устраивает встроенный в ПЗУ шрифт - это когда Вы создаете программу на своем национальном языке. Но, кроме национальных шрифтов, Вам могут потребоваться еще и тематические шрифты. Так, в зависимости от жанра разрабатываемой Вами программы, Вы можете разрабатывать шрифт с соответствующим ей стилем.

Самый простой способ для приготовления собственного шрифта - использовать соответствующий режим программы ARTSTUDIO, но, во-первых, не у всех есть этот графический редактор, а мы конечно же создаем наши книги не для тех, у кого и так все есть, а во-вторых, он не позволяет напрямую создавать символы UDG-графики.

Ниже предлагается несложная БЕЙСИК-программа, которая позволяет сгенерировать собственный символьный набор, включая и символы UDG. Программа сделана так, чтобы по-возможности исключить возможные варианты сбоя, хотя без сомнения дотошный пользователь сможет найти способ "завалить" ее. Если такое произойдет, не пугайтесь, а смело давайте команду RUN для повторного запуска. То, что Вы уже создали, не будет утрачено. Единственно только в этом случае не нажимайте клавишу "1" в главном меню программы, поскольку этим могут быть "вытерты" все созданные Вами собственные шрифты.

Программа исполнена на БЕЙСИКе, поскольку от нее не требуется скорости в работе. Скорость может потребоваться только если Вы захотите вернуться к стандартному символьному набору, "зашитому" в ПЗУ. На этот случай переброска шрифта из ПЗУ в область Вашего символьного набора в ОЗУ выполняется с помощью маленькой процедуры в машинных кодах.

Программные переменные.

r,q - адрес начала Вашего символьного набора в ОЗУ;
g$ - символьная переменная для проверки INPUT;
t$ - пункты меню;
a$ - символьная переменная для проверки INPUT;
x,y,z - переменные, используемые пр изображении символов увеличенного размера.
a - промежуточные данные, например результат ввода по INPUT;
w,c - данные для изображения символов;
f,b,v - переменные циклов FOR...NEXT.

```
 10 LET q=PEEK 23675 + 256* PEEK 23676: LET q=q-801
 20 POKE 23609,20
 40 CLEAR q
 50 LET q=PEEK 23730 + 256*PEEK 23731: LET q=q+31
 60 GO SUB 9000: GO SUB 9200
 70 GO SUB 140: IF PEEK 23681 <> 1 THEN LET g$ = "1":
    GO TO 520
 80 GO TO 400
 90 STOP

100 POKE 23606,0: POKE 23607,60
110 RETURN

140 LET r=q/256: POKE 23606,(r-INT r)*256:
    POKE 23607, INT r-1
150 RETURN

390 REM **** MENU ****
400 BORDER 7: PAPER 7: INK 2: CLS: GO SUB 100
410 PRINT BRIGHT 1; INVERSE 1; INK 1; TAB 7;
    "CHARGEN"
420 PRINT ' ' TAB 14; INK 1;"MENU"
430 PRINT ' TAB 10; INK 3; "CHARACTER SET"
```

```
440 PRINT ' "1 Сброс символьного набора." '
            "2 Посмотреть символ из ПЗУ." '
            "3 Посмотреть символ из ОЗУ."
450 PRINT   "4 Изменить символ." '
            "5 Сбросить один символ."
460 PRINT ' INK 3; TAB 5; "СИМВОЛЫ ГРАФИКИ UDG"
470 PRINT ' "6 Просмотр символов UDG" '
            "7 Изменение символов UDG"
480 PRINT INK 3; ' "8 Просмотр всех символов" '
            "9 LOAD / SAVE"
490 INK 0: PRINT#0; BRIGHT 1; INK 7: PAPER 4; TAB 5;
          "Нажмите нужную клавишу"; TAB 31; " "
500 LET g$=INKEY$: IF g$<CHR$ 49 OR g$ > CHR$ 57
    THEN GO TO 500
520 BEEP .1, CODE g$ -64
530 IF g$="1" THEN LET a=USR (q-30): POKE 23681,1:
    GO TO 400
540 IF g$="2" OR g$>"5" THEN GO SUB 100: GO TO 560
550 GO SUB 140
560 CLS: LET t$=("СИМВОЛЬНЫЙ НАБОР" AND g$<"6")+
    ("ГРАФИКА ПОЛЬЗОВАТЕЛЯ" AND (g$ = "6" OR g$ ="7"))
    + ("ПОЛНЫЙ ПРОСМОТР" AND g$ = "8") +
    ("ЗАГРУЗКА/ВЫГРУЗКА" AND  g$ = "9")
570 PRINT INK 3; AT 0,15-(LEN t$/2); t$ ' '
580 GO SUB (VAL g$)*200 + 600
600 PRINT #0; INK 1; "ENTER - вход в меню"
620 IF INKEY$ = "" THEN GO TO 620
630 IF INKEY$ = CHR$ 13 THEN BEEP .1,12: GO TO 400
650 GO TO 520
690 STOP

700 REM *** ИЗОБРАЖЕНИЕ ПЛАНШЕТА ***
710 PLOT 128,128
720 INK 1: DRAW 64,0: DRAW 0,-64: DRAW -64,0: DRAW 0,64
740 INK 0: RETURN

800 REM *** УВЕЛИЧЕНИЕ СИМВОЛА ***
810 LET z=w: LET y=128
```

```
820 INK 1: PRINT TAB 16;
830 LET x= INT (z/y): PRINT CHR$ 164;:
    IF x=1 THEN PRINT CHR$ 8;"■";:LET z=z-y
850 LET y=y/2: IF y<1 THEN INK 0: BRIGHT 0: RETURN
860 GO TO 830

1000 REM *** Просмотр символа ПЗУ ***
1010 INPUT "Какой символ ПЗУ показать?"; a$:
     LET a = CODE a$: IF a<32 OR a>127 OR LEN a$<>1
     THEN GO TO 1010
1020 LET a=(a-32)*8+15616
1030 GO SUB 700
1040 FOR f=1 TO 16: PRINT a$;" ";: NEXT f
1060 PRINT ' ' "Код:"
1070 FOR f=a TO a+7: LET w=PEEK f: PRINT ' f;
     "=";w;: GO SUB 800: NEXT f
1080 PRINT ' ': FOR f=1 TO 16: PRINT a$; " ";: NEXT f
1090 RETURN

1200 REM *** Просмотр символа из ОЗУ ***
1210 INPUT "Какой символ из ОЗУ показать?"; a$:
     LET a = CODE a$: IF a<32 OR a>127 OR LEN a$ <> 1
     THEN GO TO 1210
1220 LET a=(a-32)*8+q
1230 GO TO 1030

1400 REM *** Изменение символа ***
1420 INPUT "Какой символ изменить?"; a$
1430 LET a=CODE a$: IF a>127 OR a<32 OR LEN a$>1
     THEN GO TO 1420
1440 PLOT 128,152: GO SUB 720: LET c=(a-32)*8+q
1460 FOR f=c TO c+7: LET w=PEEK f: PRINT ' w;:
     PRINT TAB 10; " " AND f/2 = INT (f/2);a$;:
     GO SUB 800: NEXT f
1480 PLOT 128,80: GO SUB 720
1500 PRINT INK 3; ' "Новый код:";: FOR f=0 TO 7
1510 INK 2: INPUT w: IF w<0 OR w>255 OR w<>INT w
     THEN GO TO 1510
```

```
1520 POKE c+f,w: PRINT ' w; TAB 10; " " AND
     f/2 = INT(f/2);a$;: GO SUB 800
1530 NEXT f
1540 PRINT ' ' : FOR f=1 TO 16: PRINT CHR$ a; " ";:
     NEXT f
1550 RETURN

1600 REM *** Сброс символа ***
1620 INPUT "Какой символ сбросить?";a$
1630 LET a= CODE a$: IF a>127 OR a<32 OR LEN a$>1
     THEN GO TO 1620
1640 PLOT 128,152: GO SUB 720: LET c=(a-32)*8+q:
     LET a=(a-32)*8+15616
1650 FOR b=1 TO 2
1660 FOR f=0 TO 7: PRINT ';: LET w=PEEK(c+f):
     IF f=0 OR f=7 THEN PRINT TAB 0;: FOR v=1 TO 5:
     PRINT a$;" ";: NEXT v
1670 GO SUB 800: NEXT f
1680 IF b=1 THEN FOR v=0 TO 7: POKE (v+c),PEEK (v+a):
     NEXT v: PRINT AT 7,2;"Было:"; AT 10,0:
     PLOT 128,80: GO SUB 720
1690 NEXT b: PRINT AT 16,2;"Стало:"
1700 RETURN

1800 REM *** Просмотр символов UDG ***
1810 INPUT "Какой символ просмотреть?";a$: LET a=CODE a$:
     LET a=a+(79 AND a>64 AND a<91)+(47 AND a>96
     AND a<123): IF a<144 OR a>164 OR LEN a$ <> 1
     THEN GO TO 1810
1820 GO SUB 700: FOR f=1 TO 16: PRINT CHR$ a; " ";: NEXT f
1830 PRINT ' ' "Код UDG:"
1840 LET c=USR a$
1850 FOR f=0 TO 7: LET w=PEEK (f+c): PRINT ' w;: GO SUB 800
1860 NEXT f: PRINT ' '
1870 FOR f=1 TO 16: PRINT CHR$ a; " ";: NEXT f
1880 RETURN

2000 REM *** Изменение символа UDG ***
```

```
2010 INPUT "Какой символ изменить?";a$: LET a=CODE a$:
     LET a=a+(79 AND a>64 AND a<91)+(47 AND a>96
     AND a<123): IF a<144 OR a>163 THEN GO TO 2010
2020 LET c=USR a$(1): PLOT 128,152: GO SUB 720:
     FOR f=0 TO 7
2030 LET w=PEEK (f+c): PRINT ' w; TAB 10;" " AND
     f/2=INT(f/2); CHR$ a;: GO SUB 800: NEXT f
2040 PRINT INK 2;'"Новый код символа:";
2050 PLOT 128,80: GO SUB 720
2060 FOR f=0 TO 7: INK 2: INPUT w: IF w<0 OR w>255
     OR w <> INT w THEN GO TO 2060
2070 POKE (f+c),w: PRINT ' w; TAB 10; " "
     AND f/2 = INT(f/2); CHR$ a;: GO SUB 800: NEXT f
2080 PRINT ' ': FOR f=1 TO 16: PRINT CHR$ a;" ";: NEXT f
2090 RETURN

2200 REM *** Просмотр всего набора ???
2210 PRINT "Символьный набор ПЗУ:";
2220 INK 0: FOR g=1 TO 2
2240 FOR b=1 TO 6: PRINT: FOR f=0 TO 14: PRINT " "
     AND (b/2 = INT (b/2) AND f=0); CHR$ ((b*15)+f+17);
     " ";: NEXT f: NEXT b
2250 PRINT TAB 10;: FOR f=122 TO 127: PRINT CHR$ f;
     " ";: NEXT f
2260 IF g=1 THEN INK 2: PRINT ' "Символьный набор ОЗУ:";:
     GO SUB 140
2270 NEXT g
2280 GO SUB 100
2290 INK 3: PRINT ' "Символы UDG:": FOR f=144 TO 164
2300 PRINT CHR$ f;" "; CHR$ 23 + CHR$ 11 + CHR$ 0 AND f=159;
2310 NEXT f
2330 RETURN

2400 REM *** Процедуры загрузки-выгрузки ***
2410 PRINT INK 1; ' "A..Сохранить символьный набор." ' '
     "B..Сохранить символы UDG." ' ' "C..Сохранить эту
     программу."
```

```
2430 PRINT INK 2; ' ' "D. . Загрузить символьный набор. " ' '
     "E. . Загрузить символы UDG. " ' ' "F. . Загрузить любую
     программу. "
2450 LET a$=INKEY$: LET a = CODE a$:
     LET a=a-(32 AND a>96 AND a<103):
     IF a<65 OR a>70 THEN GO TO 2450
2460 LET b=3+ (1 AND a>67) + ((a-65)*2): PRINT AT b,0;
     INK 8; FLASH 1; OVER 1; TAB 7
2470 GO SUB (3000+((a-65)*100))
2480 IF INKEY$ = "n" OR INKEY$ = "N" THEN GO TO 2480
2490 RETURN

3000 SAVE "Charset" CODE q,768
3020 PRINT AT 20,10;"VERIFY ? (Y/N)": LET a$=INKEY$
3030 IF a$ = "n" THEN GO TO 3080
3040 IF a$ <> "y" THEN GO TO 3020
3050 PRINT AT 20,2; "Отмотайте ленту и включите магнитофон";
     AT 19,0:  IF a<> 67 THEN VERIFY "" CODE
3060 IF a=67 THEN VERIFY ""
3080 RETURN

3100 SAVE "U. D. G. " CODE (q+769),168
3120 GO TO 3020
3200 LET a$="y": SAVE "Chargen" LINE 1
3220 GO TO 3020

3300 PRINT AT 18,1;"Включите магнитофон";
3310 LOAD "" CODE q
3330 RETURN

3400 PRINT AT 18,1;"Включите магнитофон";
3410 LOAD "" CODE (q+769)
3430 RETURN
3500 PRINT AT 18,12; FLASH 1; "GOOD BYE"
3510 PRINT #1;"Press any key & start tape"
3520 LOAD "": RUN
8999 STOP
```

```
9000 REM *** Настройка графического символа 'U' ***
9010 RESTORE 9000: FOR f=0 TO 7: READ a:
     POKE USR "u"+ f,a
9020 NEXT f: RETURN
9030 DATA 128,128,128,128,128,128,128,255

9200 RESTORE 9200: LET r=q/256
9210 FOR f=(q-30) TO (q-30)+11: READ a: POKE f,a:
     NEXT f: RETURN
9220 DATA 17,(r-INT r)*256, INT r, 33,0,61,1,0,3,237,176,
     201
9990 STOP
9999 GO SUB 100: POKE 23609,0
```

ПРИЛОЖЕНИЕ 2

Простейший генератор шаблонов.

Спрайтами называют графические объекты, которые могут воспроизводиться на экране и перемещаться по нему, не изменяя изображения, которое там первоначально было. Поскольку манипуляции со спрайтами служат для обеспечения мультипликации, то изучение их не входит в рамки данной книги, а явится темой очередного тома - "Динамическая графика". В то же время, если от вопросов перемещения спрайтов по экрану пока отвлечься, то программу для их редактирования можно использовать и для создания статических шаблонов.

Здесь рассматривается программа для создания шаблонов изображении размером 24X21 пиксел. Такой размер достаточно удобен по целому ряду соображений. С одной стороны, это достаточно большой размер, позволяющий реализовать художественные способности при создании образов, с другой стороны, он не настолько велик, чтобы скорость операций печати шаблона стала заметно критичной для программиста. В принципе, размер шаблонов 24X24 может быть был бы чуть более удобней при программировании, но зато при размере 24X21 мы можем так организовать генератор, чтобы на экране каждый пиксел будущего спрайта изображался одним знакоместом.

Поскольку эта программа очень удобна для создания спрайтов для Ваших программ, мы воспользуемся ею же и в третьем томе нашей книги - "Динамическая графика".

Программа состоит из двух частей. Первая часть - это головной управляющий блок, написанный на БЕЙСИКе. Он приведен в Листинге_1. Вторая часть - исполнительные процедуры в машинных кодах. Они выполняют такие операции, как инвертирование изображения, реверсирование изображения, переброску картинки из рабочего буфера в отведенную для спрайтов область памяти, вызов их оттуда в буфер на редактирование и т. п. Блок машинных кодов приведен в Листинге_2 вместе с загрузчиком.

Сначала наберите машиннокодовый блок (Листинг_2). Запустите загрузчик командой RUN. Если все набрано правильно, Вы получите сообщение STOP. Если же где-то Вы сделали ошибку, то получите сообщение "??". Не сошлась контрольная сумма - следует еще раз внимательно проверить строки DATA. После того, как все будет закончено, Вы можете сохранить блок на ленте командой

SAVE "sprcode" CODE 54200,350.

Теперь можете приступать к набору БЕЙСИКа (Листинг_1). Он запускается командой RUN.

В этом редакторе Вы можете одновременно создавать (редактировать) и хранить до 10 спрайтов одновременно (обозначаются буквами от "a" до "j"). Когда с одним из них Вы работаете, он находится в рабочем буфере. По окончании работы с ним он копируется в область таблицы спрайтов - это делает команда RANDOMIZE USR 54317. Противоположное действие по вызову нужного спрайта из таблицы в буфер на редактирование исполняет команда RANDOMIZE USR 54353. Изображение на экране того, что находится в буфере, исполняет команда RANDOMIZE USR 54200.

Кроме простого рисования спрайтов, Вы можете в этой программе инвертировать изображение и реверсировать его. Инвертирование выполняется по нажатию клавиш CAPS SHIFT "I" - при этом изображение вместо "черным по белому" становится "белым по черному". Реверсировать (переворачивать) изображение можно по горизонтали (CAPS SHIFT "R") или по вертикали (CAPS SHIFT "U"). Вернуться в исходное меню можно нажатием CAPS SHIFT "Z".

Перемещение курсора по полю выполняется курсорными клавишами 5, 6, 7, 8. При этом не происходит рисования. Перемещение курсора с рисованием исполняется этими же клавишами при нажатой клавише SYMBOL SHIFT.

Редактор удобен еще и тем, что рядом с увеличенным изображением спрайта выстраивается и его изображение в натуральную величину.

Спраиты хранятся в таблице, начинающеися с адреса 54600. Поскольку каждый спраит состоит из 504 (24 X 21) элемеитов, то для него необходимы 63 (63 X 8 = 504) байта. Таким образом, вся таблица десяти спраитов занимает область памяти размером 630 баитов.

Листинг_1

Генератор (редактор) спрайтов.

```
 10 LOAD "sprcode" CODE 54200,350
100 LET a=22529: LET J=0: LET s=a: LET p=s
110 GO SUB 1200
120 GO TO 560
130 POKE p,233
140 LET h$=INKEY$
150 RANDOMIZE USR 54200
160 PAUSE 1.5
170 LET p=s: LET J = PEEK p
180 IF h$="I" THEN RANDOMIZE USR 54422:
    RANDOMIZE USR 54200
190 IF h$="R" THEN RANDOMIZE USR 54436:
    RANDOMIZE USR 54200
200 IF h$="U" THEN RANDOMIZE 54482:
    RANDOMIZE USR 54200
210 IF h$="Z" THEN GO TO 480
220 IF h$<"4" OR h$>"8" THEN GO TO 240
230 GO TO 370
240 IF h$<"%" OR h$>"(" THEN GO TO 140
250 IF h$="%" THEN LET s=s-1
260 IF h$="&" THEN LET s=s+32
270 IF h$="'" THEN LET s=s-32
280 IF h$="(" THEN LET s=s+1
290 GO SUB 1160
300 IF w=0 THEN LET s=s+1: GO TO 140
310 IF w>24 THEN LET s=s-1: GO TO 140
320 IF s<a THEN LET s=s+32: GO TO 140
330 IF s>a-1+21*32 THEN LET s=s-32: GO TO 140
340 POKE p,0
350 POKE s,130
```

```
360 GO TO 140
370 IF h$="5" THEN LET s=s-1
380 IF h$="6" THEN LET s=s+32
390 IF h$="7" THEN LET s=s-32
400 IF h$="8" THEN LET s=s+1
410 GO SUB 1160
420 IF w=0 THEN LET s=s+1: GO TO 140
430 IF w>24 THEN LET s=s-1: GO TO 140
440 IF s<a THEN LET s=s+32
450 IF s>a-1+21*32 THEN LET s=s-32
460 POKE p,56
470 POKE s,135: GO TO 140
480 PRINT AT 1,26; "SAVE "
490 PRINT AT 3,26; "a-j"
500 LET h$=INKEY$
510 IF h$<"a" OR a$>"j" THEN GO TO 500
520 LET v=CODE h$-96: POKE 23300,v
530 RANDOMIZE USR 54317
540 BEEP .4,20
550 GO SUB 1000
560 PRINT AT 1,26; "E a-j"
570 PRINT AT 3,26; "1 SAVE"
580 PRINT AT 5,26; "2 LOAD"
590 LET h$=INKEY$
600 IF h$="1" THEN GO TO 1100
610 IF h$="2" THEN GO TO 1130
620 IF h$<"a" OR h$>"j" THEN GO TO 590
630 LET v=CODE h$-96: POKE 23300,v
640 GO SUB 1000
650 RANDOMIZE USR 54353
660 PRINT AT 7,26; h$; " ";v;" "
670 BEEP .4,30
680 GO TO 130
1000 PRINT AT 1,26;"          "
1010 PRINT AT 3,26;"          "
1020 PRINT AT 5,26;"          "
1030 RETURN
1100 INPUT "SAVE "; n$: IF n$ = "" THEN GO TO 1100
```

```
1110 SAVE n$ CODE 54600,699
1120 GO TO 40
1130 INPUT "LOAD ";n$: LOAD n$ CODE: CLS
1140 GO TO 40
1150 STOP
1160 LET f=INT (s/32): LET w=s-f*32
1170 RETURN
1200 FOR k=8 TO 200 STEP 8
1210 PLOT k,8: DRAW 0,167
1220 NEXT k
1230 FOR k=8 TO 175 STEP 8
1240 PLOT 8,k: DRAW 192,0
1250 NEXT k
1260 PLOT 8,175: DRAW 192,0
1270 RETURN
```

Листинг_2

Блок рабочих процедур.

```
2000 LET b=54200: LET z=0: LET r=350: RESTORE 3000
2010 FOR k=0 TO r-1: READ a
2020 POKE (b+k),a: LET z=z+a
2030 NEXT k
2040 LET z=INT (((z/r)-INT (z/r))*r)
2050 READ a: IF a<>z THEN PRINT "??": STOP

3000 DATA  33, 250, 212, 17, 1
3001 DATA  88, 14, 3, 213, 237
3002 DATA  83, 246, 212, 6, 21
3003 DATA  237, 91, 246, 212, 197
3004 DATA  6, 8, 26, 254, 0
3005 DATA  40, 4, 254, 130, 32
3006 DATA  6, 55, 203, 22, 195
3007 DATA  224, 211, 167, 203, 22
3008 DATA  19, 16, 235, 229, 42
3009 DATA  246, 212, 1, 32, 0
```

```
3010 DATA   9. 34. 246. 212. 225
3011 DATA   193. 35. 16. 212. 209
3012 DATA   62. 8. 131. 95. 13
3013 DATA   32. 197. 33. 250. 212
3014 DATA   17. 123. 72. 6. 3
3015 DATA   213. 197. 14. 3. 6
3016 DATA   8. 213. 126. 18. 20
3017 DATA   35. 16. 250. 209. 62
3018 DATA   32. 131. 95. 121. 61
3019 DATA   254. 1. 32. 5. 6

3020 DATA   5. 79. 24. 233. 6
3021 DATA   8. 79. 254. 0. 32
3022 DATA   226. 193. 209. 19. 16
3023 DATA   215. 201. 205. 64. 212
3024 DATA   34. 248. 212. 237. 91
3025 DATA   248. 212. 33. 250. 212
3026 DATA   1. 63. 0. 237. 176
3027 DATA   201. 58. 4. 91. 71
3028 DATA   33. 72. 213. 17. 63
3029 DATA   0. 167. 237. 82. 25

3030 DATA   16. 253. 201. 205. 64
3031 DATA   212. 34. 248. 212. 17
3032 DATA   1. 88. 14. 3. 213
3033 DATA   237. 83. 246. 212. 6
3034 DATA   21. 237. 91. 246. 212
3035 DATA   197. 6. 8. 126. 167
3036 DATA   203. 39. 245. 48. 5
3037 DATA   62. 0. 195. 120. 212
3038 DATA   62. 56. 18. 241. 19
3039 DATA   16. 239. 229. 42. 246

3040 DATA   212. 1. 32. 0. 9
3041 DATA   34. 246. 212. 225. 193
3042 DATA   35. 16. 214. 209. 62
3043 DATA   8. 131. 95. 13. 32
```

```
3044 DATA   199, 201, 33, 250, 212
3045 DATA   6, 63, 62, 255, 174
3046 DATA   119, 35, 16, 249, 24
3047 DATA   39, 33, 250, 212, 6
3048 DATA   63, 197, 126, 1, 0
3049 DATA   8, 167, 203, 31, 203

3050 DATA   17, 16, 250, 113, 193
3051 DATA   35, 16, 239, 6, 21
3052 DATA   33, 250, 212, 17, 36
3053 DATA   213, 78, 26, 119, 121
3054 DATA   18, 35, 19, 16, 247
3055 DATA   33, 250, 212, 205, 87
3056 DATA   212, 201, 33, 14, 213
3057 DATA   17, 250, 212, 6, 3
3058 DATA   197, 213, 229, 6, 10
3059 DATA   78, 26, 119, 121, 18

3060 DATA   43, 19, 16, 247, 225
3061 DATA   209, 1, 21, 0, 84
3062 DATA   93, 19, 9, 193, 16
3063 DATA   230, 24, 213, 0, 0
3064 DATA   0, 0, 0, 0, 255
3065 DATA   255, 255, 0, 0, 190
3066 DATA   190, 190, 190, 190, 190
3067 DATA   190, 190, 190, 190, 128
3068 DATA   254, 254, 254, 254, 254
3069 DATA   254, 254, 254, 0,0

3070 DATA   190, 0, 0, 0, 0
```

ПРИЛОЖЕНИЕ 3

Организация каналов пользователя для печати нестандартными шрифтами.

Прежде чем переити непосредственно к теме, вспомним о том, какие системные переменные участвуют в организации канальной информации в "Спектруме".

Системная переменная CHANS - "адрес канала данных". Ее адрес - 5C4FH (23631). В том случае, если отсутствует "БЕТА-диск" интерфеис, значение этой системной переменной равно 5CB6H (23734). Если "БЕТА-диск" подключен, то эта величина больше на 112 байтов и равна, таким образом, 5D26H (23846). Если просмотреть содержимое (дамп) памяти с указанного адреса (в случае магнитофонного варианта - 23734 - это сразу же за таблицей системных переменных), то можно увидеть, как располагается канальная информация сущеествующих каналов.

После включения компьютера этих каналов - 4. При этом на информацию о каждом канале отводится по 5 байтов памяти (см. Листинг_1).

Листинг_1

Адрес (дес.)	Адрес (шестн.)	Содержимое (шестн.)	Значение	
23734	5CB6	F4	PRINT=09F4H	Процедура ПЗУ
23735	5CB7	09		"PRINT-OUT"
23736	5CB8	A8	INPUT=10A8H	Процедура ПЗУ
23737	5CB9	10		"KEY-INPUT"
23738	5CBA	4B	Символ "K" - канал клавиатуры	
23739	5CBB	F4	PRINT=09F4H	Процедура ПЗУ
23740	5CBC	09		"PRINT-OUT"
23741	5CBD	C4	INPUT=15C4H	Процедура ПЗУ
23742	5CBE	15		RST 8
23743	5CBF	53	Символ "S" - канал экрана	

Продолжение Листинга_1

Адрес (дес.)	Адрес (шестн.)	Содержимое (шестн.)	Значение	
23744	5CC0	81	PRINT=09F4H	Процедура ПЗУ
23745	5CC1	0F		"PRINT-OUT"
23746	5CC2	C4	INPUT=15C4H	Процедура ПЗУ
23747	5CC3	15		RST 8
23748	5CC4	52	Символ "R" - внутреннии канал	
23749	5CC5	F4	PRINT=09F4H	Процедура ПЗУ
23750	5CC6	09		"PRINT-OUT"
23751	5CC7	C4	INPUT=15C4H	Процедура ПЗУ
23752	5CC8	15		RST 8
23753	5CC9	50	Символ "P" - каиал принтера	
23754	5CCA	80	маркер <КОНЕЦ>	
23755	5CCB	...	Начало Бейсик-программы - (PROG)	

Из приведеиной в Листинге_1 информации видим, что канал "K" - клавиатура - обслуживается двумя процедурами: вывод (PRINT) и ввод (INPUT). Их адреса соответственно равны 09F4H и 10A8H. Ввод - это опрос клавиш, а вывод - это печать в двух нижних строках экрана. Хотя физически нижние две строки принадлежат экрану, но по архитектуре "Спектрума" в режиме ввода данных или редактирования Бейсик-строки нижние две строки относятся вовсе не к экрану, а к клавиатуре. Там мы видим результат нажатия на клавиши.

Канал "S" - экран, а точнее, его осиовная часть, т.е. верхние 22 строки - обслуживается только процедурои PRINT (той же, что и для клавиатуры). А ввод (INPUT) для экрана невозможен - вместо процедуры INPUT для экрана задан адрес 15C4H - это вызов процедуры ПЗУ для печати сообшения об ошибке с кодом перехвата 12H, что означает: "Invalid I/O device".

Каиал "R" - является внутренним каналом "Спектрума", он служит для регенерации памяти и его практическое использование требует особой осторожности. Касаться этои темы мы пока не будем.

Канал "P" - принтер - изначальио имеет тот же адрес процедур PRINT, что и экран. Так же, как и для экрана INPUT - невозможен. Но в том случае, когда производится инициализация интерфейса принтера, вместо адреса 09F4H для процедуры PRINT задается другой адрес - в зависимости от используемого интерфейса. Например, в том случае, если используется интерфейс ZX-LPRINT III, то после инициализации его командои LPRINT, для канала принтера устанавливается адрес процедуры PRINT=0EFCH.

Заканчивается канальная информация маркером конца - 80H. Далее, с адреса 5CCBH (23755) располагается Бейсик-программа. Этот адрес указан в системиои переменной PROG.

Для организации дополнительных пользовательских каналов, позволяющих выполнять, скажем, вывод на экран при помоши процедур, отличных от традиционной, можно пойти следующим путем. Надо, используя процедуру ПЗУ, отодвинуть всю область БЕЙСИК-системы на 5 баитов вверх для каждого дополнительного канала. При этом должны быть соответствующим образом скорректированы все системные переменные - в первую очередь - PROG. Для этого в ПЗУ имеется специальиая процедура 1655H (05717), которая используется, когда иадо вставить БЕЙСИК-строку между уже имеюшимися строками, раздвинув их. Эта процедура иазывается MAKE_ROOM. Перед тем, как ее вызывать, задают в регистровой паре HL адрес, указывающий на точку, следующую за местом вставки, а в регистровои паре BC задают длину этои "вставки".

Не вдаваясь в подробности этого мероприятия, можем заметить, что измеиение PROG крайне нежелательно, так как часто в иулевои строке расположены блоки кодов, которые не могут быть загружены в иное место памяти или к иим происходит обращение из Бейсика без учета установленного зиачения PROG. Всю "головиую боль", которую доставляет изменеиие PROG, сполна могут оце-

нить владельцы "БЕТА-диск" интерфейсов. Уж кто-кто, а они-то сполна это прочувствовали, адаптируя программы под диск.

Между тем, существует способ, который позволяет создавать до трех дополнительных пользовательских каналов, не изменяя PROG. При этом "Спектрум" работает как обычно, совершенно ничего не подозревая об "обмане".

Но, прежде, чем перейти к изложению этого приема, надо вспомнить еще об одной системной переменной - STRMS - точнее считать ее даже и не переменной, а таблицей длиной 38 байтов. Ее адрес - 5C10H (23568) - см. Листинг_2.

Листинг_2

Адрес (дес.)	Адрес (шестн.)	Содержимое (шестн.)	Поток	Канал	Рассчет адреса
23568	5C10	01	FD	"K"	23734-1+1=23734
23569	5C11	00			
23570	5C12	06	FE	"S"	23734-1+6=23739
23571	5C13	00			
23572	5C14	0B	FF	"R"	23734-1+11=23744
23573	5C15	00			
23574	5C16	01	00	"K"	23734-1+1=23734
23575	5C17	00			
23576	5C18	01	01	"K"	23734-1+1=23734
23577	5C19	00			
23578	5C1A	06	02	"S"	23734-1+6=23739
23579	5C1B	00			
23580	5C1C	10	03	"P"	23734-1+16=23750
23581	5C1D	00			
23582	5C1E	00	---	---	
23583	5C1F	00			
...	...	...	...	...	
23604	5C34	00	---	---	
23605	5C35	00			

То, какой канал к какому потоку подключен, определяется следующим образом. К значению, на единицу меньшему, чем системная переменная CHANS, прибавляется величина смещения, содержащаяся в соответствующей данному потоку паре байтов системной переменной STRMS. Например, для потока 02 видим величину смещения, равную 6. Он подключен к тому каналу, 5-байтный блок канальной информации которого расположен с адреса 23739 (см. расчет). По Листингу_1 можем определить, что это - "экран". Потоки 00 и 01 - оба подключены к клавиатуре. Действительно, выполнив PRINT #0; или PRINT #1; - печать будет произведена в двух нижних строках, а PRINT #2; - в главной части экрана.

Из Листинга_2 видим, что используется потоков всего 7 - с FD по 03 (с минус третьего по третий), то есть по 2 байта на каждый поток - 14 байтов. Остальное пространство переменной STRMS не используется, так как не задано других потоков. При организации новых потоков в этой системной переменной будут задаваться новые пары байтов для каждого нового потока. Места столько, что можно задать целых 12 новых потоков! Конечно, столько новых потоков нам может не потребоваться, однако вот это свободное место оказалось удобно использовать для хранения совсем другой информации - канальной, как бы в продолжение той, которая начинается с адреса 23734 (заданного в CHANS, Листинг_1). Делается такой "обман" следующим образом.

Предположим, что мы хотим задать новые каналы для клавиатуры, экрана и принтера. Пусть новая процедура печати для клавиатуры и экрана имеет адрес AB00H, новая процедура опроса клавиш - адрес AC80H, а новая процедура печати для принтера - адрес 5B20H. Тогда информация, расположенная в системной переменной STRMS будет иметь вид, представленный в Листинге_3.

Листинг_3

Адрес (дес.)	Содержимое (шестн.)	Поток (шестн.)	Канал	Рассчет адреса
23568	5C10	01	FD	"K" 23734-1+1=23734
23569	5C11	00		

```
23570    5C12    06      FE      "S"     23734-1+6=23739
23571    5C13    00
23572    5C14    0B      FF      "R"     23734-1+11=23744
23573    5C15    00
23574    5C16    01      00      "K"     23734-1+1=23734
23575    5C17    00
23576    5C18    01      01      "K"     23734-1+1=23734
23577    5C19    00
23578    5C1A    06      02      "S"     23734-1+6=23739
23579    5C1B    00
23580    5C1C    10      03      "P"     23734-1+16=23750
23581    5C1D    00
23582    5C1E    6F      04      "K"     23734-1+65391-
23583    5C1F    FF                          -65536=23588
23584    5C20    74      05      "S"     23734-1+65396-
23585    5C21    FF                          -65536=23593
23586    5C22    79      06      "P"     23734-1+65401-
23587    5C23    FF                          -65536=23598

Далее располагается канальная информация, продолжая Листинг_1.
23588    5C24    00      PRINT=AB00H       Процедура
23589    5C25    AB                      "PRINT-NEW"
23590    5C26    80      INPUT=AC80H       Процедура
23591    5C27    AC                    "KEY-INPUT-NEW"
23592    5C28    4B      Символ "K" - канал клавиатуры
23593    5C29    00      PRINT=AB00H       Процедура
23594    5C2A    AB                      "PRINT-NEW"
23595    5C2B    C4      INPUT=15C4H       Процедура ПЗУ
23596    5C2C    15                        RST 8
23597    5C2D    53      Символ "S" - канал экрана
23598    5C2E    20      PRINT=5B20H       Процедура
23599    5C2F    5B                      "PRINT-NEW"
23600    5C30    C4      INPUT=15C4H       Процедура ПЗУ
23601    5C31    15                        RST 8
23602    5C32    50      Символ "P" - канал принтера
23603    5C33    00
23604    5C34    00      3 неиспользуемых байта.
23605    5C35    00
```

Здесь надо сказать, что совсем не обязательно задавать новые процедуры и для клавиатуры, и для экрана, и для принтера. Это могут быть три любые новые процедуры, например, три новые (разные) процедуры печати для экрана, которые подключаются к потокам #4...#6. Кроме того, совсем не обязательно, чтобы их было именно три. Можно задать, например, всего одну новую процедуру печати (например, для потока #4). Тогда ячейки 23584...23587 и 23593...23605 просто не будут использоваться.

Инициализация, то есть операция подключения нового канала (новой процедуры) к новому потоку может быть выполнена, например, прямо из БЕЙСИКа, изменением при помощи POKE значений ячеек в соответствии с конкретными требованиями, аналогично примеру, приведенному в Листинге_3. При помощи БЕЙСИКа инициализация выполняется в демонстрационных программах плотной и качественной печати, приведенных в разделах 2.8.1 и 2.8.2 данной книги.

После того, как инициализация выполнена, вновь подключенные каналы (или канал) будут действовать до тех пор, пока не будет выполнена команда NEW, которая опять задает исходное подключение каналов, как после рестарта компьютера.

Для инициализации новых каналов (подключения их к новым потокам) можно воспользоваться и более универсальным вариантом в машинных кодах, который приводится ниже, в Листинге_4. Это процедура инициализации трех новых пользовательских каналов. Процедура может быть использована и теми, кто работает с магни-тофоном и теми, кто работает с БЕТА-диском, - ее можно загружать в любое место памяти, запуская с адреса загрузки. Для примера дизассемблер этой программы дан с адреса FEDBH, но с одинаковым успехом она будет работать и в других адресах.

Листинг_4

```
Адрес Маш.код  АССЕМБЛЕР          Комментарий

FEDB  ED5B4F5C LD   DE, (#5C4F) ;В DE - значение CHANS.
FEDF  21255C   LD   HL, #5C25   ;Величина для расчета смещения.
FEE2  ED52     SBC  HL,DE       ;Расчет смещения канальной ин-
```

```
                                        ;Формации.
FEE4  221E5C    LD    (#5C1E),HL        ;Запись значения смешения.
FEE7  010500    LD    BC,#0005          ;Расчет следующего
FEEA  09        ADD   HL,BC             ;значения смешения.
FEEB  22205C    LD    (#5C20),HL        ;Запись следующего значения.
FEEE  09        ADD   HL,BC             ;Повторение операции еще раз
FEEF  22225C    LD    (#5C22),HL        ;всего - для трех потоков.
FEF2  CD7C00    CALL  #007C             ;Привязка к адресу
FEF5  3B        DEC   SP                ;загрузки; После этих
FEF6  3B        DEC   SP                ;действии в HL будет
FEF7  E1        POP   HL                ;адрес, следующий за
FEF8  011000    LD    BC,#0010          ;этой процедурой, т.е.
FEFB  09        ADD   HL,BC             ;в этом примере - FF05H.
FEFC  11245C    LD    DE,#5C24          ;Адрес места назначения - 23588
FEFF  010F00    LD    BC,#000F          ;Длина перебрасываемого блока -
                                        ;15 байтов.
FF02  EDB0      LDIR                    ;Переброска канальной информа-
                                        ;ции.
FF04  C9        RET                     ;Возврат из инициализирующей
                                        ;процедуры.
-------------------------------------------------------------------------
FF05  C4                                ;Здесь задана канальная
FF06  15                                ;информация, в формате,
FF07  C4                                ;как в Листинге_1, т.е.
FF08  15                                ;по 5 байтов на канал
FF09  4B                                ;(всего для трех каналов -
FF0A  C4                                ;- 15 байтов.)
FF0B  15                                ;Все значения 15C4H
FF0C  C4                                ;должны быть заменены
FF0D  15                                ;на адреса реальных
FF0E  53                                ;процедур, обслуживающих
FF0F  C4                                ;новые каналы.
FF10  15                                ;Изменены должны быть
FF11  C4                                ;и литеры "K", "S" и "P",
FF12  15                                ;обозначающие
FF13  50                                ;новые каналы.
```

Уважаемый читатель!

Мы искренне благодарны Вам за то, что Вы смогли уделить некоторую часть своего времени нашей книге. Мы надеемся, что и первый том серии "Элементарная графика" Вы тоже уже прочитали.

Если основной целью "Элементарной графики" было дать Вам представление о технологии исполнения графических экранов из БЕЙСИКа и из машинного кода и снабдить Вас необходимыми знаниями для дальнейшего самообразования путем исследования кода фирменных программ, то по прочтении этой книги Вы безусловно можете самостоятельно начинать практические эксперименты с машинной графикой.

Достаточно ли того, что Вы прочитали для самостоятельной разработки конкурентоспособных программ? Возможно, да, но скорее всего Вам не обойтись без анимации (мультипликации) графических изображений и в этом Вам поможет следующий том нашей серии - "Динамическая графика".

"ИНФОРКОМ" сейчас работает над широким спектром новых книг для владельцев персонального компьютера системы "Синклер". В их числе и книги для тех, кто хорошо знает свой компьютер и нуждается в углубленной информации и книги для тех, кто только что начал делать первые шаги с компьютером. Если Вы подписаны на наше фирменное издание - "ZX-РЕВЮ", то всегда своевременно узнаете о наших новых работах. Если же Вы не являетесь нашим постоянным читателем, мы Вам рекомендуем периодически (один раз в 3 - 4 месяца) посылать нам запрос с вложенным заполненным конвертом. Вы получите наш текущий БЛАНК-ЗАКАЗ с описанием распространяемой нами литературы.

Наш адрес: 121019, Москва, Г-19, а/я 16.