Г. Блэнд

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ БЕЙСИК В СТАНДАРТЕ MSX

Г. Блэнд

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ БЕЙСИК В СТАНДАРТЕ MSX

Перевод с английского Ю.Е. Поляка
Предисловие и дополнение А.В. Гиглавого

МОСКВА
"ФИНАНСЫ И СТАТИСТИКА"
1989

MSX PROGRAMMING

Graham Bland

PITMAN

ББК 24.4.1
Б-709

Блэнд Г.
Б-709 Основы программирования на языке Бейсик в стандарте MSX: Пер. с англ.: Предисл. и дополн. А.В.Гиглавого.– М.:Финансы и статистика, 1989. – 208 с.:ил.

ISBN 5-279-00253-4.

В книге известного английского автора изложен вводный курс программирования на Бейсике в MSX-стандарте, ориентированном на 8-разрядные микроЭВМ ("Ямаха" и др.). Параллельно вводятся основные понятия языка, принципы устройства ЭВМ, элементы культуры программирования. Текст иллюстрирован практичными программами. Рассмотрены расширенные графические (управление "спрайтами") и звуковые возможности MSX-Бейсика.

Б $\frac{2404010000-029}{010(01)-89}$ 128–89

ББК 24. 4. 1

ISBN 5-279-00253-4 (СССР)
ISBN 0-273-02302-0 (Великобритания)

ПРЕДИСЛОВИЕ К РУССКОМУ ИЗДАНИЮ

История возникновения и распространения стандарта MSX, охватывающего архитектуру, конструкцию и программные средства бытовых компьютеров минимальной стоимости, является поучительным примером предприимчивости небольшой группы инженеров и коммерсантов из разных стран. Возглавлял группу Казухико Ниси, японский предприниматель, занимавший в 1984 г. пост вице-президента токийского филиала крупнейшей американской фирмы по производству программных продуктов для персональных ЭВМ "Майкрософт".

Базой для разработки стандарта стала конструкция бытового компьютера гонконгской фирмы "Спектравидео". К.Ниси убедил разработчиков этого компьютера Фокса и Вайсса в том, что предложенная ими конструкция может стать "платформой" для унифицированного программного обеспечения бытовых компьютеров; в сочетании с дешевой технологией крупнейших японских фирм идея такой унификации обещала резко повысить уровень конкурентоспособности японских бытовых компьютеров на рынке стран Западной Европы и США.

Последующие события показали, что проникновение компьютеров MSX на рынок США не состоялось в силу двух важнейших причин: обострения "торговой войны" между Японией и США в области бытовой электроники и высокой насыщенности американского рынка дешевыми компьютерами предыдущего поколения, среди которых выделялись компьютеры фирм "Коммодор" и "Атари". В Японии, Южной Корее и ряде стран Западной Европы (Голландия, Испания, Италия, ФРГ) бытовые компьютеры MSX получили широкое распространение. Для пользователей этих компьютеров издаются журналы и бюллетени новинок, созданы ассоциации.

В октябре 1984 г. было объявлено о разработке дисковой операционной системы MSX DOS, которая стала неотъемлемой частью стандарта MSX. По набору функций система MSX DOS была предельно приближена к наиболее популярной ОС для восьмиразрядных персональных ЭВМ – CP/M (она нашла применение на некоторых отечественных микроЭВМ и описана в ряде переводных публикаций). По файловой структуре на гибких магнитных дисках система MSX DOS совместима с операционной системой MS DOS/PC DOS для 16-разрядных персональных

ЭВМ с архитектурой IBM PC. Если учесть, что диалект языка Бейсик MSX представляет собой расширение диалекта GW-Бейсик, предложенного ранее фирмой "Майкрософт" для IBM PC, то можно сделать вывод о достаточно близком "родстве" этих двух моделей персональных компьютеров. Любопытно, что во второй половине 80-х годов упомянутая выше фирма "Спектравидео" создала компьютер-гибрид IBM PC и MSX.

Более 15 фирм Азии и Западной Европы выпустили в середине 80-х годов несколько миллионов бытовых компьютеров MSX и MSX-2 (вторая версия стандарта имеет существенно расширенные графические возможности). В СССР используется более 20 тыс. компьютеров такого типа; в основном они входят в состав комплектов учебной вычислительной техники (КУВТ), закупленных у японской фирмы "Синдзидайся" для учебных заведений.

Крайне ограниченный условиями контракта состав документации и программных средств для КУВТ вызвал недоумение у пользователей в СССР. Впрочем, эти неудобства возымели неожиданные для многих последствия: уже к 1986 г. в нашей стране был создан значительный фонд инструментальных, учебных и игровых программ для КУВТ, и сегодня оснащенные этой техникой кабинеты информатики стали основой для подготовки школьников, студентов в педагогических институтах и преподавателей в институтах усовершенствования учителей. Среди языков программирования, использованных авторами данных программ, наряду с языком машинных команд (языком Ассемблера) наиболее широкое распространение получили языки Бейсик, Паскаль и Си.

Предлагаемая советскому читателю книга Г.Блэнда послужит хорошим введением в технику программирования на Бейсике MSX для тех, кто имеет минимальный опыт программирования на микроЭВМ и программируемых калькуляторах. Среди распространенных в СССР диалектов Бейсика для ЭВМ массового применения (СМ ЭВМ, "Электроника", ДВК, "Агат" и др.) большинство пока составляют диалекты, не поддерживающие машинную графику. В то же время необходимо подчеркнуть, что реализация Бейсика для бытовых компьютеров серии БК во многом следует правилам стандарта MSX. Учитывая отмеченную ранее близость диалектов MSX и GW-Бейсика, можно рекомендовать эту книгу и начинающим, которые приступают к изучению основ программирования для профессиональных персональных ЭВМ,— таких, как ЕС1841/42, "Искра" и "Нейрон".

Компьютеры серии MSX обладают весьма разнообразными возможностями в сравнении с другими компьютерами аналогичной стоимости. Гибкость архитектуры MSX позволяет решать целый ряд задач, в частности:

* Обработка текстов и передача данных. В распоряжении пользователя имеются разнообразные программы текстовых редакторов для различных языков (английского, французского, немецкого, арабского, русского, японского и ряда других). С помощью интерфейса RS232C можно связать компьютеры через модемы и канал телефонной линии.

* Создание и исполнение музыкальных произведений. Применение модуля звуковых эффектов и необходимых программных средств превращает компьютер в электромузыкальный синтезатор с дополнительными возможностями ввода, редактирования, хранения на магнитных носителях и вывода мелодий на принтер.

* Машинная графика и обработка изображений. Программные средства компьютера,— такие, как программа "Painter", реализованная в учебных компьютерах "КУВТ Ямаха MSX-2" образца 1987 г.,— позволяют создавать весьма сложные рисунки, которые можно записывать на дискеты и выводить на принтер. Кроме того, для компьютеров MSX-2 разработана аппаратура ввода изображений, поступающих от внешнего источника видеосигналов; эти изображения в цифровой форме могут быть обработаны и записаны на дискеты.

* Игры. Для компьютеров MSX разработаны и тиражируются сотни игровых программ, записанных на магнитных носителях или в специальных кассетах с постоянной полупроводниковой памятью.

* Изучение основ программирования. Язык программирования Бейсик был разработан специально для изучения основ программирования. Впоследствии он занял ведущее место среди языков, применяемых на персональных компьютерах. MSX-версия Бейсика, или MSX-Бейсик, оставаясь доступной для начинающих, предоставляет в то же время все средства обращения к аппаратуре компьютера, необходимые специалисту.

* Выполнение научно-технических расчетов. В интерпретаторе языка MSX-Бейсик имеется библиотека стандартных программ математических функций, благодаря которой многие расчеты могут быть выполнены без помощи высокопроизводительных ЭВМ. Программы, которым не требуется для работы большой объем оперативной памяти и высокое быстродействие процессора, могут выполняться "в домашних условиях". Эти возможности дополнены описанными ранее возможностями обработки текстов и передачи данных.

Такое разнообразие применений обеспечивается особенностями архитектуры и конструкции компьютеров, отвечающих требованиям стандарта MSX. Разъемы на корпусе компьютера позволяют подключать модули дополнительной оперативной памяти (RAM), кассеты с программами, записываемыми в ПЗУ (ROM), модули электромузыкальных синтезаторов и адаптеры различных периферийных устройств.

В любом случае трудно представить себе ситуацию, когда пользователь персонального компьютера обходится без знания основ программирования. Программирование является важной составной частью компьютерной грамотности; только овладение программированием позволяет использовать все возможности компьютера.

В настоящей книге излагаются основные сведения о компьютере MSX. Изучение этого материала позволит начинающим оценить весь спектр возможностей персонального компьютера с тем, чтобы выбрать некоторые из них для более детального рассмотрения. Книга построена по схеме последовательного усложнения; многочисленные примеры помогают проверить на практике полученные знания.

Автор стремился избежать свойственного многим пособиям по программированию "перечислительного" стиля изложения. Разумеется, построенные подобным образом справочники удобны для профессионалов; однако начинающим трудно ориентироваться в таком материале. Поэтому весь справочный аппарат книги вынесен в приложения.

Диапазон рассматриваемых тем оказывается весьма широким — от изучения правил работы с клавиатурой до приемов программирования графических и звуковых эффектов. Для закрепления материала в руководстве подробно разобраны 118 программ. Такой подход применительно к Бейсику оправдал себя во многих опубликованных ранее пособиях.

Практика использования Бейсика в школьных и вузовских курсах информатики уже позволила создать устойчивые методики начального обучения программированию. Книга Г.Блэнда вполне может быть рекомендована и как дополнительное пособие в этих курсах, и как пособие для самостоятельного изучения Бейсика владельцами бытовых компьютеров.

Канд. техн. наук А.В.Гиглавый

ПРЕДИСЛОВИЕ

Книга посвящена основам программирования на Бейсике в системе MSX. Предполагается, что читатель в какой-то мере знаком со своим компьютером,— например набирал тот или иной текст на клавиатуре.

В книгу включены примеры программ для демонстрации обсуждаемых в ней особенностей языка. По мере возможности эти программы написаны таким образом, чтобы их можно было практически использовать (в том числе для их же проверки). Так, в гл. 5 описана некоторая функция и показано, как с ее помощью можно разместить текст в центре экрана дисплея.

Читатели могут заметить определенный уклон в сторону графики, которой посвящены две важнейшие главы. По-видимому, именно графическим возможностям (великолепно реализованным в MSX-Бейсике) с их интересными, динамичными, зрелищными эффектами домашние, да и прочие компьютеры обязаны большей частью своей популярности. Однако доступная широкому читателю информация о работе с видеопамятью и видеопроцессором довольно скудна (если вообще существует). С целью дополнить ее в книгу включена гл.9.

В книге не делалось никаких попыток излагать вопросы программирования в машинных кодах. Это вполне оправдано назначением книги и тем фактом, что она главным образом посвящена программированию на Бейсике. В то же время в приложении излагается механизм включения в MSX-Бейсик-программу подпрограмм в машинных кодах.

MSX-Бейсик представляет собой идеальное введение в Бейсик для большого числа компьютеров. Беглый анализ версии BASICA, разработанной для персональных компьютеров IBM, показывает, что MSX-Бейсик, в сущности, идентичен языку 16-разрядной машины. Знакомство со звуковыми операторами MSX-Бейсика позволит программистам при желании перейти к версии GW–BASIC 2.0, поставляемой рядом фирм, выпускающих 16-битные компьютеры, в том числе Compaq, DEC, Hitachi, Olivetti, Sanyo, Wang и многими другими в разных странах мира.

В заключение мне хотелось бы выразить признательность Джанет Моррисон за полезную критику и технические советы при подготовке книги.

Эта книга посвящается юной Эмме Луизе Фенвик, которая пока находится в блаженном неведении о существовании компьютеров и, надеюсь, останется в этом неведении еще несколько лет.

Грехем Блэнд
Октябрь 1985

НОТАЦИЯ

При описании команд и функций MSX-Бейсика в книге принята следующая нотация:

* Ключевые слова набираются заглавными буквами и не могут быть опущены.
* Символы, заключенные в угловые скобки < >, не могут быть опущены.
* Символы, заключенные в квадратные скобки [ ], являются необязательными.
* Из списка символов, разделенных знаком | , требуется выбрать только один.
* Повторяющиеся символы обозначаются многоточием (...).

Все прочие знаки пунктуации должны использоваться так, как показано в тексте.

Глава 1. ВВОДНЫЕ ЗАМЕЧАНИЯ

Компьютер представляет собой электронное устройство, предназначенное для решения разнообразных задач — от детских игр до выполнения сложных расчетов. Столь широкий диапазон действий объясняется способностью компьютера распознавать определенный набор инструкций, а затем выполнять их. Последовательность таких инструкций называется ***программой***.

Вопреки распространенному мнению о компьютерах они ни сверхразумны, ни глупы. Это всего лишь машины, которые скрупулезно и буквально исполняют посылаемые им команды. Автор команд, программист, обязан четко и ясно сообщить компьютеру порядок выполнения задания. Язык MSX-Бейсик создан именно для облегчения связи между человеком и машиной.

Эта глава знакомит читателя с MSX-Бейсиком и элементами компьютеров MSX. Мы начнем с обзора основных компонентов компьютерной системы.

ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР (ЦП)

Центральный процессор — основной элемент любой компьютерной системы. Он способен воспринимать некоторый набор инструкций и выполнять их. Инструкции представляют собой последовательности определенных чисел, содержащие ***машинные коды***.

К функциям ЦП относятся выполнение всех вычислений, а также управление работой большинства элементов компьютера. Во всех компьютерах MSX центральным процессором является микропроцессор Zilog Z80.

ПАМЯТЬ

Память предназначена для хранения информации. Информация может иметь вид списка инструкций, образующих машинную программу, или представлять собой данные, обрабатываемые программой. Нам необходимо рассмотреть два основных типа памяти.

1. Оперативное запоминающее устройство (ОЗУ) – часть памяти, содержащая лишь информацию, записанную после последнего включения машины.

2. Постоянное запоминающее устройство (ПЗУ) – область памяти, предназначенная только для чтения, но не для записи. В ПЗУ хранится предварительно "зашитая" в него информация (как правило, программы). Эта информация сохраняется и при отключении электропитания.

УСТРОЙСТВА ВВОДА И ВЫВОДА

Устройства ввода используются для снабжения центрального процессора информацией. Важнейшее из них, без сомнения, клавиатура. Другими примерами такого типа устройств могут служить джойстик (координатная рукоятка), световое перо, сенсорный экран.

Для вывода информации необходим ***монитор*** (обычный бытовой телевизор либо специально разработанное устройство). Еще одно устройство вывода – ***принтер*** (печатающее устройство).

Некоторые виды оборудования могут играть роль как источника, так и приемника информации. Наиболее дешевое и популярное устройство ввода-вывода – это кассетный накопитель на магнитной ленте. Вместо него можно использовать накопитель на диске, обладающий высоким быстродействием, но более дорогой.

БИТЫ И БАЙТЫ

Основным элементом памяти цифрового компьютера является *бит* (от английского выражения binary digit – двоичная цифра). Бит может принимать одно из двух значений: 0 либо 1. Для представления информации используется набор из восьми бит, который называется ***байтом***. Процессор Z80 считается восьмибитным (или восьмиразрядным), поскольку он оперирует с величинами, сгруппированными по восемь бит.

С помощью байта можно представить 256 различных чисел (от 0 до 255) в двоичной системе счисления. В отличие от обычной десятичной системы в двоичной системе числа записываются по основанию 2. Вместо разрядов единиц, десятков, сотен, тысяч и т.д. в ней выделяются единицы, двойки, четверки, восьмерки и т.д. Приведем несколько примеров двоичных чисел:

Двоичная форма	Десятичная форма
100	4
11110001	241
1101	13

Байт – не единственное наименование группы бит. Часто используются и такие обозначения:

Полубайт	4	бита
Слово	16	бит
Двойное слово	32	бита

АДРЕСАЦИЯ ПАМЯТИ

Ваш компьютер, как следует из паспортных данных, имеет память объемом в 64К, 48К, 32К или, возможно, 16К. Здесь К обозначает 10-ю степень числа 2, т.е. 1024. Таким образом, для компьютера в 64К объем памяти составляет

1024 * 64 = 65536 байт.

Каждый байт памяти представляет собой единственный элемент, которому можно присвоить имя для адресации. Это имя записывается в виде числа. N-й по порядку ячейке памяти присваивается номер N–1. Так, 65536-я ячейка имеет номер 65535.

ИНТЕРПРЕТАТОР MSX-БЕЙСИКА

Любой MSX-компьютер имеет ПЗУ емкостью 32К, где хранится специальная программа в машинных кодах, называемая ***интерпретатором MSX***. Сразу же при включении компьютера центральный процессор считывает и выполняет команды интерпретатора.

Интерпретатор начинает работу с генерации сообщения об авторстве программы – "копирайта". Кроме того, выполняется ряд тестов для компьютера. В результате вы получаете сообщение о размере обьема ОЗУ, свободного для ваших программ и данных. Эти размеры могут быть различными, но для большинства систем (32–64К) составляют 28815 байт.

Главная роль интерпретатора состоит в том, что он позволяет вам вводить, редактировать и исполнять Бейсик-программы. Эти программы переводятся интерпретатором в машинные коды, после чего они могут восприниматься и выполняться центральным процессором.

Появившееся на экране монитора приглашение "Ok" сигнализирует о том, что интерпретатор готов к работе и ожидает инструкций. Они могут быть представлены в ***командном*** или в ***непосредственном*** режиме. Существует довольно много различных инструкций, большинство из них связано с вводом и редактированием программ.

Попытайтесь ввести команду Бейсика CLS (после чего следует нажать клавишу RETURN). Если компьютер работает нормально, то по этой команде экран будет очищен и на нем появится приглашение "Ok". CLS – непосредственная команда: интерпретатор опознает фразу CLS (CLear Screen – очистить экран) и выполняет необходимые действия немедленно. А теперь попробуйте ввести команду CLD. Интерпретатор выдаст фразу "Syntax error" (синтаксическая ошибка). CLD не входит в тот ограниченный словарь фраз, который распознается интерпретатором. С подобными сообщениями о синтаксических ошибках вам, возможно, придется сталкиваться чаще всего в процессе программирования.

Непосредственные команды выполняются только один раз. Для повторного выполнения команды ее нужно набрать заново.

ПРОГРАММЫ

Программа 1.1 переводит длины отрезков, выраженные в дюймах, в метрическую систему. Наберите на клавиатуре эту программу и выполните ее. Запуск вычислений осуществляет команда RUN, предписывающая интерпретатору начать трансляцию и выполнение программы.

Программа 1.1

```
10 REM *
20 REM * Перевод в метрическую систему
30 REM *
40 CLS
50 PRINT "Введите длину в дюймах"
60 INPUT INCHES
70 METRES = INCHES * .0254
80 PRINT "Длина в метрах примерно равна"
90 PRINT METRES
100 END
```

Программа должна запрашивать у пользователя значение длины в дюймах, а затем выдавать результат в метрах. На этом примере мы можем сделать некоторые общие выводы о программах и рассмотреть ряд операторов и команд Бейсика.

Номера строк. Каждая строка программы должна начинаться с номера. Наличие или отсутствие номера позволяет отличить оператор Бейсика, являющийся частью программы, от непосредственной команды. Номера строк должны быть целыми числами в диапазоне от 0 до 65529. Как правило, номера соседних строк отличаются на 10.

Интерпретатор выполняет строки программы по порядку номеров. Если ввести строки программы в обратном порядке, интерпретатор переставит их в порядке возрастания. При вводе строки с уже имеющимся номером последняя из введенных строк уничтожает предыдущую и запоминается как часть программы.

В одной строке можно размещать несколько операторов, отделяя их друг от друга двоеточием. Так, несложную программу

```
10 INPUT INCHES
20 PRINT METRE
```

можно записать следующим образом:

```
10 INPUT INCHES : PRINT METRE
```

Комментарии. Оператор REM указывает, что следующий за ним текст программной строки не транслируется и не исполняется. Этот оператор позволяет размещать полезные комментарии в любом месте вашей программы. Вместо служебного слова REM можно пользоваться апострофом " ` ".

ОПЕРАТОР PRINT

Команда PRINT применяется для вывода информации на экран монитора. Например, в программе 1.1 выводится фраза "Введите длину в дюймах". Все, что заключено между двойными кавычками, называется *строкой*. Попробуйте выполнить такой пример:

```
10 PRINT "ПРИВЕТ";
20 PRINT "ПРИВЕТ"
```

и вы получите следующий результат:

```
ПРИВЕТПРИВЕТ
```

Как правило, после вывода фрагмента информации очередная порция, предназначенная для печати, располагается на следующей строке. При использовании символа ";" в качестве разделителя печати переход к новой строке отменяется, и очередная порция информации выводится вслед за предыдущей. Если же разделителем служит запятая, результат будет иным. Например, оператор

```
PRINT "ПРИВЕТ", "ПРИВЕТ"
```

выводит два слова "ПРИВЕТ" на расстоянии в 14 символов.

Слово PRINT можно заменить своего рода стенографическим символом – вопросительным знаком. Интерпретатор преобразует вопросительный знак в слово PRINT сразу при вводе строки программы.

ПРЕКРАЩЕНИЕ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Оператор END обозначает конец программы. Этот оператор включать в текст необязательно, поскольку MSX-Бейсик выполняет программу до тех пор, пока не будут исчерпаны операторы, подлежащие трансляции и исполнению.

Оператор STOP приостанавливает выполнение программы. Работу можно возобновить со следующей после STOP строки, набрав команду CONT (continue – продолжить).

Нажатием клавиши STOP вы останавливаете выполнение программы до повторного нажатия той же клавиши. Если нажать одновременно клавиши CTRL и STOP, выполнение программы прекратится. Этот полезный механизм позволяет вашей программе при необходимости прервать работу без каких-либо сбоев и аварий.

ИНДИКАЦИЯ ТЕКСТА ПРОГРАММЫ

По команде LIST текст программы выводится из памяти на экран монитора. Если у вас есть принтер, этот текст можно напечатать на бумаге с помощью команды LLIST. Ниже приведены примеры различных вариантов использования команд вывода.

LIST .	Печать текущей строки
LIST –40	Печать текста программы с начала до строки 40 включительно
LIST 40 –	Печать с 40-й строки и до конца

УДАЛЕНИЕ СТРОК

Если в программу вводится строка с определенным номером, причем предложение с таким номером уже имеется в памяти, то старый текст будет стерт. Команда DELETE исключает из программы строки и блоки строк следующим образом:

DELETE 100	Удаление строки с номером 100
DELETE –100	Удаление всех строк по 100-ю включительно
DELETE 50 –100	Удаление строк с 50-й по 100-ю включительно

Команда NEW полностью очищает память от программных строк.

ПЕРЕНУМЕРАЦИЯ ПРОГРАММ

Команда RENUM изменяет нумерацию строк программы. Формат этой команды таков:

RENUM [[<новый номер>][,<старый номер>][,<приращение>]]

Ниже даются примеры.

RENUM	Всем строкам программы присваиваются номера начиная с 10-го с шагом 10
RENUM 40	Всем строкам начиная с 40-й присваиваются номера с шагом 10
RENUM ,,5	Всем строкам присваиваются номера начиная с 10-го с шагом 5

АВТОМАТИЧЕСКАЯ НУМЕРАЦИЯ СТРОК

Если при вводе программы применить команду AUTO, интерпретатор будет автоматически создавать номера строк. Всякий раз при вводе строки в начале следующей строки появляется очередной номер. По команде

AUTO 100,5

генерируются номера строк начиная с 100-го с шагом 5.

ИЗМЕНЕНИЕ ФОРМАТА ЭКРАНА

При работе с текстами можно пользоваться двумя вариантами экранного формата. Один из них позволяет работать с 24 строками, каждая из которых не длиннее 40 символов, другой содержит те же 24 строки, но длиной не свыше 32 символов.

Существенно большая гибкость достигается с помощью команды WIDTH, которая очищает экран и устанавливает предельное число символов в строке в соответствии с заданием. Так, команда

WIDTH 20

допускает до 20 знаков в строке.

Расположенные в нижней части экрана служебные символы можно удалить командой KEY OFF, а при необходимости вновь установить командой KEY ON. Исключение этой служебной информации позволяет освободить 24-ю строку экрана для программ.

КОМАНДА SCREEN

MSX-Бейсик позволяет работать с четырьмя типами экранов. Два из них являются графическими и обсуждаются в гл. 8. Выбор типа экрана осуществляется по команде SCREEN с указанием соответствующего номера:

0 – текстовой экран размером 40 * 24;
1 – текстовой экран размером 32 * 24;
2 – графический экран с высоким разрешением;
3 – графический экран с низким разрешением.

Приведем полное синтаксическое описание этой команды.

SCREEN <режим>[,<размер спрайта>][,<звук клавиши>] [,<скорость ленты>][,<параметры печати>]

Параметр <звук клавиши> позволяет включать и отключать звуковое сопровождение (щелчок), которое вызывает каждый удар по клавишам.

```
SCREEN ,,0    (выключение звука)
SCREEN ,,1    (включение звука)
```

ИЗМЕНЕНИЕ ЦВЕТА

Цвета текста, фона и границы экрана можно изменять командой COLOR. Ниже описывается диапазон допустимых цветов:

0	прозрачный	8	красный
1	черный	9	розовый
2	средне-зеленый	10	темно-желтый
3	светло-зеленый	11	желтый
4	темно-синий	12	темно-зеленый
5	светло-синий	13	васильковый
6	бордовый	14	серый
7	голубой	15	белый

Команда имеет следующий формат:

COLOR <основной текст>,<фон>,<граница>.

Например, по команде

COLOR 15,1

текст выводится белым шрифтом на черном фоне.

ФУНКЦИОНАЛЬНЫЕ КЛАВИШИ

Каждой функциональной клавише можно сопоставить строку длиной до 15 символов. Если вы хотите, чтобы при нажатии клавиши F1 на экране появлялось слово ПРИВЕТ, а при нажатии F10 – MSX, то нужно воспользоваться командами:

```
KEY 1,   "ПРИВЕТ"
KEY 10,  "MSX"
```

Познакомиться с текущими значениями всех функциональных клавиш можно, набрав на клавиатуре KEYLIST.

ЗАПИСЬ И СЧИТЫВАНИЕ ПРОГРАММ

Программу можно записать на магнитную ленту одним из двух способов. Команда CSAVE записывает программу во ***внутреннем*** формате. Это наиболее быстрый способ. Чтобы убедиться в правильности записи, следует перемотать ленту и выполнить команду CLOAD?. Последняя сравнивает программу, считываемую с ленты, с программой, находящейся в данный момент в памяти, и проверяет их соответствие. По команде CLOAD программу можно загрузить с ленты в память.

Команда SAVE записывает программу в формате кода ASCII. Этот способ медленнее по сравнению с предыдущим, но имеет ряд преимуществ. Если программа записана с помощью команды SAVE, она может загружаться с ленты и выполняться автоматически.

Команда LOAD считывает с ленты программы, снабженные заголовками. Если выполнить команду

LOAD "CAS:",R

то первая программа в кодах ASCII, обнаруженная на ленте, начнет загружаться в память и автоматически выполняться. Аналогичный эффект дает команда

RUN "CAS:"

где "CAS:" – кассетный накопитель. Здесь вы впервые встречаетесь с примером ***мнемонического имени устройства*** (см. гл. 6). Другое мнемоническое имя, "LPT:", обозначает принтер. Если выполнить команду SAVE с именем "LPT:", результат будет таким же, как и при выполнении команды LLIST.

Изменение скорости записи. Обычно программы записываются на ленту со скоростью 1200 бит/с, или Бод. Располагая очень хорошим кассетным магнитофоном, скорость записи можно удвоить. Для этого в операторе SCREEN значение параметра <скорость записи> следует установить равным двум:

```
SCREEN ,,2
```

Теперь программа будет записываться со скоростью 2400 Бод.

Для увеличения скорости записи можно также воспользоваться командой CSAVE. Чтобы, например, записать на ленту программу "FRED" со скоростью 2400 Бод, нужно ввести команду в таком формате:

```
CSAVE "FRED",2
```

На этом мы заканчиваем краткий обзор некоторых команд Бейсика. Многие из них применяются скорее для манипуляций с программами, чем в программировании как таковом. В гл. 2 вы познакомитесь с основными понятиями программирования.

СВОДКА СЛУЖЕБНЫХ СЛОВ

```
CLS
COLOR <основной текст>,<фон>,<граница>
WIDTH <ширина строки экрана>
SCREEN <режим>[,<размер спрайта>][,<звук клавиши>]
[,<скорость ленты>][,<параметры печати>]
REM
PRINT [<выражение>][<разделитель>][<выражение>]...
END
STOP
CONT
RUN [<номер строки>]
RUN "<имя программы>"
LIST [<начальный номер>]-[<конечный номер>]
LLIST [<начальный номер>]-[<конечный номер>]
DELETE [<начальный номер>]-[<конечный номер>]
RENUMBER [[<новый номер>][,<старый номер>][,<приращение>]]
AUTO [<номер строки>][,<приращение>]
KEY <номер функциональной клавиши>,<строка текста>
KEY ON | OFF | LIST
CSAVE [<имя программы>][,<1 | 2>]
CLOAD?
CLOAD [<имя программы>]
SAVE <имя программы>
LOAD <имя программы>[,R]
```

Глава 2. ЭЛЕМЕНТАРНЫЙ MSX-БЕЙСИК

В настоящей главе рассматривается ряд важных элементов языка: данные и типы данных, операторы, ветвления. Владение этим материалом необходимо для перехода к более сложным понятиям.

ДАННЫЕ И ТИПЫ ДАННЫХ

Программам для работы требуется информация. Информация, используемая в компьютере, обозначается термином ***данные***.

Существуют два основных источника получения данных. Информацию можно ввести с клавиатуры (с помощью команд типа INPUT), с кассеты, с помощью джойстика или любого иного устройства ввода. С другой стороны, данные могут уже существовать в памяти компьютера обычно как фрагмент программы или как результат выполнения предыдущих действий.

Данные подразделяются на классы. Часть информации сохраняет свое значение при всех обстоятельствах. Число сантиметров в одном метре, длина выражения "MSX", значение ускорения свободного падения – это примеры ***констант***.

Величины же, подобные суточной температуре воздуха, длине произвольного слова в словаре или биржевому индексу, непрерывно изменяются, им нельзя приписать постоянное значение. Такие данные называются ***переменными***.

Элемент данных может также принадлежать к одному из различных ***типов данных***. Диапазон разрешенных типов данных для констант шире, чем для переменных. Ниже приводится полный список допустимых в MSX-Бейсике типов данных.

1.	Целые	(целые числа)
2.	Действительные	(числа с десятичной точкой)
3.	Строковые	(символьные данные)
4.	Двоичные	(числа по основанию 2)
5.	Шестнадцатиричные	(числа по основанию 16)
6.	Восьмиричные	(числа по основанию 8)

Рассмотрим более подробно, как перечисленные выше типы данных приписываются константам и, когда это возможно, переменным, а также их достоинства и типичные неудобства при использовании.

Целые числа. Целая константа представляет собой целое число, записанное без десятичной точки. При записи таких констант к числу приписывается справа знак процента %. Допустимый диапазон для целых величин – от –32768 до 32767. Наряду с константами можно использовать и целые переменные; их признаком также служит символ %.

Целые величины часто применяются в тех случаях, когда дробная часть не нужна и даже нежелательна, например при подсчете. Главное неудобство при работе с ними заключается в том, что набор выражаемых ими чисел относительно мал. Однако для хранения в памяти целой величины требуется всего два байта.

Действительные числа. Это более полезный тип данных в основном из-за того, что он позволяет работать с дробями. В такой форме можно представить широкий спектр чисел:

$9.9999999999999 * 10^{62}$ – максимально допустимое число.
10^{-63} – минимальное число.

В MSX-Бейсике существуют различные уровни точности представления действительных чисел, которые влияют на объем памяти, необходимый для хранения информации:

* одинарная точность, означающая, что у константы или переменной должны храниться шесть старших разрядов. При этом для каждого числа необходимо четыре байта памяти;
* двойная точность, позволяющая увеличить точность до 14 значащих цифр. Следовательно, для хранения числа с двойной точностью требуется больший обьем памяти – 8 байт.

Различают также два способа представления действительных чисел. Наибольшей популярностью пользуется первый из них – представление с фиксированной точкой, например:

```
1692.34    0.000729    -36    -511.05
```

Другой способ – представление с плавающей точкой. Он особенно полезен при работе с очень большими и очень малыми числами. Примеры чисел с плавающей точкой:

```
1.654E-6    9.681E-24    6.823392394D-35    -23.545D60
```

Здесь и E и D обозначают операцию умножения на 10 в нужной степени. D указывает, что число хранится с двойной точностью.

Уровень точности константы или переменной легко определить по виду записи: величины с одинарной точностью заканчиваются символом !, а с двойной – символом # .

В MSX-Бейсике величины и переменные по умолчанию подразумеваются имеющими тип данных с двойной точностью.

Строки. ***Строковую константу*** образуют любые символы, заключенные между двойными кавычками. Такими символами могут быть буквы, цифры, пробелы и любые другие знаки, которые можно напечатать. При определении константы имеет значение каждый символ между кавычками. Например, "MSX-Бейсик" — совсем не та же константа, что " MSX-Бейсик ", поскольку во втором случае в начале и конце кавычки отделены от текста пробелами.

Единственный символ, который нельзя включать в строковую константу, — это сами кавычки. Так, строка

"Они кричали "Ура!" все хором"

вызовет сообщение об ошибке. К сожалению, при работе со строковыми переменными обойти это ограничение нельзя.

Другим ограничением является длина строки символов: она не должна превышать 255 знаков. По умолчанию в MSX-Бейсике для всех строк первоначально в памяти отводится пространство в 200 символов. Ниже даны примеры строковых констант.

"Бигл – летчик" "4634.377" "2 умножить на 2"

Имя строковой константы должно оканчиваться знаком доллара $.

Двоичные константы. Целые числа можно записать в двоичной системе. Двоичная константа представляет собой последовательность двоичных цифр, начинающуюся знаками &B. Примеры положительных двоичных констант:

&B11111111	десятичное число 255
&B1111	15
&B0111111111111111	32767

Простейший способ получить двоичную форму положительного десятичного числа состоит в последовательном делении на два. Рассмотрим этот алгоритм на примере числа 137.

Операция деления			Результат	Остаток
137	/	2	68	1
68	/	2	34	0
34	/	2	17	0
17	/	2	8	1
8	/	2	4	0
4	/	2	2	0
2	/	2	1	0
1	/	2	0	1

Как только результат последнего деления становится равным нулю, двоичное представление исходного числа можно образовать из всех промежуточных остатков. Остаток от первого деления становится наименьшим

по значению (крайним справа) разрядом, затем записываются остальные и т.д. до старшего разряда, который образован последним остатком. Таким образом, двоичным эквивалентом десятичного числа 137 является 10001001.

Отрицательные двоичные числа представить в виде констант несколько сложнее. В MSX-Бейсике отрицательные числа хранятся в дополнительном двоичном коде. Чтобы преобразовать отрицательное число в его дополнительную двоичную форму, следует:

* записать двоичное представление соответствующего положительного числа;
* заменить все нули единицами, а единицы — нулями;
* к полученному двоичному числу прибавить единицу. При этом важно помнить, что имеется в виду полное 16-битное представление исходного числа, поэтому необходимо учитывать все ведущие нули. Покажем последовательность шагов для получения дополнительного двоичного представления числа –15:

1.	&B0000000000001111	=	15_{10}
2.	&B1111111111110000	после инверсии всех битов	
3.	&B1111111111110001	=	-15_{10}

Приведем еще три примера отрицательных чисел:

1.	&B1000000000000001	=	-32767_{10}
2.	&B1111111100000001	=	-255_{10}
3.	&B1111111111111111	=	-1_{10}

Каждое двоичное число, содержащее единицу в старшем, 16-м, разряде (2^{15}), в MSX-Бейсике автоматически рассматривается как отрицательное. Поэтому 16-й разряд называется ***знаковым***.

Двоичные константы широко используются при работе с таким оборудованием, как, например, плата звукового генератора, где значения определенных битов могут быть весьма существенны.

Шестнадцатиричные константы. Каждое значащее целое число можно также представить в виде шестнадцатиричной константы. Числа записываются по основанию 16, при этом десятичное число 16 имеет вид 10. Десятичные числа от 10 до 15 обозначаются буквами A–F. Запись шестнадцатиричных констант начинается символами &H, например:

&HFF (255) &H20 (32) &HFFFF (–1)

Отрицательные числа записываются в дополнительном двоичном коде.

Существует простой способ перевода двоичных чисел в шестнадцатиричные. Он заключается в следующем.

1. Число разбивается на группы по 4 бита, начиная справа.

2. Каждая четырехбитовая группа заменяется ее десятичным эквивалентом. Для чисел, больших или равных 10, нужно пользоваться соответствующими буквами (A–F).

Продемонстрируем описанный процесс на примере десятичного числа 449, записанного первоначально в двоичном виде:

1.	&B000111000001			–	исходное двоичное число
2.	0001	1100	0001	–	после разбиения на "четверки"
3.	1	12	1	–	после перевода в десятичную систему
4.		&H1C1		–	результат: шестнадцатиричное число

Шестнадцатиричная форма в основном применяется при программировании в машинных кодах именно благодаря свойству компактного представления двоичных чисел.

Восьмиричные константы. Это числа, записанные по основанию 8. Их включение в алфавит языка может показаться несколько странным. Большинство программистов считает, что шестнадцатиричных и двоичных чисел для работы вполне достаточно. Восьмиричные числа включены для полноты описания.

Восьмиричные константы при записи начинаются символами &O. На них распространяются те же ограничения, что и для шестнадцатиричных и двоичных констант.

Перевод двоичного числа в восьмиричную форму выполняется так же, как и в шестнадцатиричную, только группы при разбиении числа должны содержать по три бита вместо четырех. Посмотрите, как переводится в восьмиричную систему число 449.

1.	&B000111000001				–	исходное двоичное число
2.	000	111	000	001	–	после разбиения на "тройки"
3.	0	7	0	1	–	после перевода в десятичную систему
4.		&O701			–	результат: восьмиричное число

На этом мы завершаем рассмотрение типов данных, допустимых в MSX-Бейсике.

ИМЕНА ПЕРЕМЕННЫХ

В MSX-Бейсике имеется четыре типа переменных: вещественные, целые, строковые и массивы. Массивы будут детально обсуждаться в гл. 6.

Понятие переменной обозначает область памяти, где могут храниться данные любого из перечисленных типов. Переменная, описанная как вещественная с двойной точностью, может содержать лишь вещественные числа с двойной точностью, строковая переменная – только строковые данные и т.д.

Каждой переменной присваивается имя. Все имена переменных должны начинаться с буквы, за которой могут следовать, если угодно, до 252 символов. Последним символом может быть один из *знаков описания типа* – # ,!, % или $. Необходимо отметить, что в имени переменной существенными являются лишь первые два символа. Так, названия "VOLKSWAGEN" и "VOLUME" в MSX-Бейсике будут иметь одинаковое внутреннее представление "VO".

Имена переменных не могут содержать в своем составе служебных слов MSX-Бейсика. В качестве имен недопустимы ни PRINT, ни PREMIUM (здесь содержится служебное слово REM). Некоторые на первый взгляд вполне приемлемые имена переменных могут тем не менее вызывать сообщения об ошибках. К их числу относится слово LOCATION, включающее фрагмент LOC. LOC не используется в MSX-Бейсике, но обозначает команду получения номера текущей записи в дисковой системе MSX. Список зарезервированных слов MSX-Бейсика, которые нельзя включать как фрагменты в имена переменных, приведен в приложении 3.

ОПЕРАТОРЫ MSX-БЕЙСИКА

Вполне естественно, в MSX-Бейсике предусмотрен набор операторов для выполнения арифметических действий. Они представлены в табл. 2.1.

Таблица 2.1

Оператор	Пример	Пояснение
+	X + Y	Сложение X и Y
–	X – Y	Вычитание Y из X
MOD	X MOD Y	Целый результат деления X на Y
*	X * Y	Умножение X на Y
/	X / Y	Деление X на Y
^	X ^ Y	Возведение X в степень Y

Еще один оператор для выполнения действий над числами – унарный минус (слово "унарный" означает, что требуется только один операнд). Этот оператор заменяет число на противоположное ему по знаку. Так, если A=10 и выполняется выражение A=–A, то A приобретает значение –10.

Операции над строками выполняет лишь один оператор ***конкатенация*** (присоединение), обозначаемый знаком "+". Например, результатом действия

"Красный " + "цветок"

является строка

"Красный цветок".

Арифметическим выражением называется любая последовательность операторов, констант, переменных и скобок. Они могут быть простыми

A + B – C – 12

или более сложными, например:

(A * (–B)/ MOD 2) ^ (2–1).

Заметим, что скобки здесь используются так же, как и в обычных алгебраических выражениях. В отсутствие скобок последовательность операторов выполняется в таком порядке: возведение в степень, инверсия, ум-

ножение и деление, действия по модулю (MOD), сложение и вычитание. Несколько однотипных операторов вычисляются слева направо. Применение скобок нарушает естественный порядок следования операторов.

ПРИСВАИВАНИЕ ЗНАЧЕНИЙ ПЕРЕМЕННЫМ

Простейший способ задать переменной ее значение состоит в применении оператора присваивания. Он записывается в одной из двух форм.

LET <имя переменной> = <выражение>

или

<имя переменной> = <выражение>

Вторая форма короче и удобнее. Знак "=" называется ***оператором присваивания***. Переменная слева от него приобретает значение выражения в правой части.

В роли выражения может употребляться как простая переменная, так и арифметическое выражение. В программе 2.1 показаны разные способы выполнения присваивания переменным их значений.

Программа 2.1

```
10 A = 2
20 B = 10
30 PRINT "A = ": A
40 A = A^B
50 PRINT "A = ": A
```

Еще одна возможность присвоить переменным значения связана с применением оператора INPUT. Простейший формат оператора ввода данных:

INPUT X

Компьютер в ответ на это печатает вопросительный знак и ждет, пока не будет набрано число и нажата клавиша RETURN. Переменная X получит значение, равное введенному числу. При выполнении оператора

INPUT X,Y,Z

ожидается ввод трех чисел, разделенных запятыми. Если ввести лишь одно число, компьютер выдаст двойной вопросительный знак, указывающий на то, что требуется дополнительная информация.

В оператор INPUT могут также включаться строковые выражения. Допускается, в частности, такой формат:

INPUT "Пожалуйста, введите число"; X

В результате выполнения этого оператора на экране появляется введенный текст (без кавычек) и вопросительный знак, после чего компьютер будет ожидать ввода значения величины X. Строка символов, фигурирующая здесь, обычно называется подсказкой.

Присваивание значений с помощью операторов READ и DATA. Оператор READ применяется для выборки констант из списка данных в операторе DATA и присвоения этих значений переменным. В качестве примера рассмотрим программу 2.2, в результате выполнения которой переменным A,B и C будут присвоены значения соответственно –45, 540 и 1.7.

Программа 2.2

```
10 REM *
20 REM * Операторы READ и DATA
30 REM *
40 READ A
50 READ B
60 READ C
70 PRINT "A = ";A
80 PRINT "B = ";B
90 PRINT "C = ";C
100 DATA –45,540,1.7
```

Строки также могут содержаться в операторе DATA, например:

```
DATA Строки,могут,содержаться
```

Как видите, в операторе DATA строковые переменные не должны заключаться в кавычки. Исключением является случай, когда строка содержит пробелы:

```
DATA "Строка содержит пробелы"
```

При обсуждении операторов READ и DATA следует также остановиться на команде RESTORE. После выполнения RESTORE очередное значение, запрашиваемое оператором READ, выбирается из самого начала списка данных в операторе DATA. Пользуясь оператором RESTORE с указанием номера строки, например RESTORE 90, можно добиться того, чтобы очередной элемент данных читался оператором READ из оператора DATA, находящегося на указанной строке или ниже ее. Действие оператора RESTORE иллюстрируется программой 2.3.

Программа 2.3

```
10 REM *
20 REM * Оператор RESTORE
30 REM *
40 RESTORE 130
50 READ A$
60 PRINT A$
70 READ A$
80 PRINT A$
90 RESTORE
100 READ A$
```

```
110 PRINT A$
120 DATA "Первый оператор DATA"
130 DATA "Второй оператор DATA"
140 DATA "Третий оператор DATA"
```

Операторы READ и DATA весьма полезны, особенно при отладке программ. Можно сформировать ряд операторов DATA, содержащих различные тестовые данные. Операторы DATA, хранящие списки данных, могут располагаться в любом месте программы.

Команда SWAP. Как следует из ее названия, команда SWAP обеспечивает обмен значений двух величин, хранящихся в виде переменных. К двум переменным можно применять команду SWAP только в том случае, если предварительно им присвоены определенные значения. Общий формат команды:

```
SWAP A,B
```

Разумеется, обмениваться значениями могут лишь величины с одинаковыми типами данных. Эта команда весьма полезна. Для достижения аналогичного результата без ее использования требуется целая программа, подобная программе 2.4.

Программа 2.4

```
10 A=10:B=5
20 PRINT "A= ";A
30 PRINT "B= ";B
40 C=A
50 A=B
60 B=C
70 PRINT "После обмена"
80 PRINT "A= ";A
90 PRINT "B= ";B
```

ОПЕРАТОР GOTO

Оператор GOTO изменяет порядок выполнения программы. Если компьютер выполняет команду

```
50 GOTO 100
```

то сразу после нее начнут исполняться операторы, записанные начиная с 100-й строки. GOTO представляет собой оператор ***безусловного перехода***.

Характерный пример применения GOTO — это организация циклов, которые дают возможность группе операторов программы повторяться многократно. В программе 2.5 показан бесконечный цикл. Выполнение программы будет продолжаться до тех пор, пока вы не нажмете клавиши CONTROL–STOP.

Программа 2.5

```
10 PRINT "Бесконечный цикл"
20 GOTO 10
```

Реальную мощь оператора GOTO можно почувствовать, если применять его в сочетании с операторами принятия решения, например если последовательность операторов должна выполняться лишь в течение того времени, пока значение некоторой величины меньше 10. В следующем разделе вводятся важные операторы и предложения, позволяющие на основе проверки условия и принятия решения выполнять определенные действия.

УСЛОВНЫЕ ОПЕРАТОРЫ И ВЕТВЛЕНИЯ

Условные переходы отличаются от безусловных. При их использовании порядок выполнения операторов программы зависит от результата проверки некоторого ***условия***. Рассмотрим такое предложение: "Если возраст человека больше или равен 18 годам, то он имеет право голосовать". Ключ к соответствующим предложениям Бейсика дается служебными словами IF и THEN. Общий формат оператора IF...THEN:

IF <условие> THEN <список операторов>|<номер строки>

[ELSE <список операторов>|<номер строки>]

либо

IF <условие> GOTO <номер строки>

[ELSE <список операторов>|<номер строки>]

Если параметр ELSE отсутствует, компьютер проверяет условие и выясняет, истинно ли оно. При положительном результате проверки выполняется набор операторов, следующих за словом THEN, либо происходит переход к другой строке. В противном случае будет выполняться оператор, следующий за оператором IF...THEN.

При наличии параметра ELSE и ложности проверяемого условия выполняются операторы, следующие за словом ELSE, либо происходит переход к соответствующему номеру строки. Заметим, что слово ELSE должно находиться в программе на той же строке, что и IF...THEN.

Операторы отношений. Операторы, позволяющие проверять условия, содержащиеся в IF...THEN, называются ***операторами отношений***. Они представлены в табл. 2.2.

Таблица 2.2

Оператор	Отношение
=	Равенство (не путать с присваиванием)
<>	Неравенство
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно

Возвращаясь к примеру с голосованием, мы можем проверить условие "больше или равно 18" с помощью такого оператора:

```
30 IF AGE>=18 THEN PRINT "Вы можете голосовать"
```

Сложные условные операторы. В тех случаях, когда перед принятием решения требуется проверить выполнение сразу двух условий, применяются сложные условные операторы. Рассмотрим утверждение: "Если возраст ребенка больше или равен 5 годам и в то же время меньше или равен 16 годам, то ребенок обязан посещать школу". Ребенок должен ходить в школу, если оба эти условия выполняются одновременно. В Бейсике такая проверка реализуется следующим образом:

```
30 IF AGE>=5 AND AGE<=16 THEN PRINT "Вы обязаны
посещать школу"
```

Служебное слово AND представляет собой пример ***логического оператора.*** Если одно из проверяемых условий не выполнено (например, AGE=56 или AGE=1), значит, результат проверки отрицательный, и строка печататься не будет. Другим логическим оператором является OR. Здесь результат окажется отрицательным, если оба условия ложны.

Результаты выполнения логических операторов удобно анализировать с помощью ***таблиц истинности.*** В такой таблице приводятся все возможные варианты проверки условий. Выполнение условия (истина) обозначается цифрой 1, невыполнение (ложь) – 0. Ниже приведена таблица истинности для оператора AND.

A	B	A AND B
0	0	0 (ложь)
1	0	0 (ложь)
0	1	0 (ложь)
1	1	1 (истина)

Оператор NOT является унарным логическим оператором, результатом выполнения которого является "ложь", если проверяемое условие истинно, и наоборот. Например, в выражении

```
30 IF AGE>=18 AND NOT (C$="Россия") THEN
PRINT "Вы можете голосовать"
```

текст будет печататься только в том случае, когда C$ имеет любое значение, кроме "Россия", и значение возраста не меньше 18.

Из всех логических операторов чаще всего употребляются AND, NOT и OR. Полный список логических операторов приводится в табл. 2.3, а их таблицы истинности – в приложении 4. Как и для прочих операторов, для них существует определенный порядок выполнения, который можно изменить применением скобок.

Таблица 2.3

Оператор	Действие
NOT	Инверсия
AND	Логическое И
OR	Логическое ИЛИ
XOR	Исключающее ИЛИ
EQV	Эквивалентность
IMP	Импликация (следование)

В программе 2.6 мы встречаемся с многочисленными проверками условий и разветвлениями. Сначала пользователь программы вводит день и месяц своего рождения в виде двух целых чисел (операторы 110 и 130). Например, 19 июня вводится как 6 и 19. Программа проверяет допустимость введенных данных (месяц не может иметь значение, скажем, 33, а день – 46), затем определяет зодиакальное созвездие, соответствующее введенной дате, и выводит информацию, как показано на рис. 2.1.

Программа 2.6

```
10 REM ************************
20 REM *                      *
30 REM * Программа "Гороскоп" *
40 REM *                      *
50 REM ************************
60 CLS
70 SCREEN 0 : WIDTH 38
80 REM *
90 REM * Запрос даты рождения
100 REM *
110 INPUT "Месяц рождения";M
120 IF M<1 OR M>12 THEN BEEP : GOTO 110
130 INPUT "Число";D
140 IF D<0 OR D>31 THEN BEEP : GOTO 130
150 PRINT
160 IF (M=3 AND D>=21) OR (M=4 AND
D<=19) THEN 310
170 IF (M=4 AND D>=20) OR (M=5 AND
D<=19) THEN 340
180 IF (M=5 AND D>=20) OR (M=6 AND
D<=20) THEN 370
190 IF (M=6 AND D>=21) OR (M=7 AND
D<=22) THEN 400
200 IF (M=7 AND D>=23) OR (M=8 AND
D<=21) THEN 430
210 IF (M=8 AND D>=22) OR (M=9 AND
D<=22) THEN 460
220 IF (M=9 AND D>=23) OR (M=10 AND
```

```
D<=22) THEN 490
230 IF (M=10 AND D>=23) OR (M=11 AND
D<=21) THEN 520
240 IF (M=11 AND D>=22) OR (M=12 AND
D<=21) THEN 550
250 IF (M=12 AND D>=22) OR (M=1 AND
D<=21) THEN 580
260 IF (M=1 AND D>=22) OR (M=2 AND
D<=19) THEN 610
270 IF (M=2 AND D>=20) OR (M=3 AND
D<=20) THEN 640
280 REM *
290 REM * Выводимая информация
300 REM *
310 PRINT "ARIES"
320 S$="Овен" : P$="Марс"
330 GOTO 660
340 PRINT "TAURUS"
350 S$="Телец" : P$="Марс"
360 GOTO 660
370 PRINT "GEMINI"
380 S$="Близнецы" : P$="Меркурий"
390 GOTO 660
400 PRINT "CANCER"
410 S$="Рак" : P$="Луна"
420 GOTO 660
430 PRINT "LEO"
440 S$="Лев" : P$="Солнце"
450 GOTO 660
460 PRINT "VIRGO"
470 S$="Дева" : P$="Меркурий"
480 GOTO 660
490 PRINT "LIBRA"
500 S$="Весы" : P$="Венера"
510 GOTO 660
520 PRINT "SCORPIO"
530 S$="Скорпион" : P$="Марс"
540 GOTO 660
550 PRINT "SAGITARIUS"
560 S$="Стрелец" : P$="Юпитер"
570 GOTO 660
580 PRINT "CAPRICORN"
590 S$="Козерог" : P$="Сатурн"
600 GOTO 660
610 PRINT "AQUARIUS"
620 S$="Водолей" : P$="Сатурн"
630 GOTO 660
640 PRINT "PISCES"
650 S$="Рыбы" : P$="Юпитер"
```

```
660 PRINT
670 PRINT "Знак Зодиака : ";S$
680 PRINT "Планета      : ";P$
```

```
Месяц рождения 6
Число 19

GEMINI

Знак Зодиака    :          Близнецы
Планета         :          Меркурий
```

Рис. 2.1. Результат работы программы 2.6

Сравнение строк. Операторы отношений можно использовать для сравнения строк символов. Но смысл понятий "меньше" или "больше" для строк не столь очевиден, как для чисел. Например, при выполнении сравнения значение "ALGY" оказывается меньшим, чем "algy". Чтобы понять причину этого, нужно иметь в виду, что каждому символу MSX-Бейсика соответствует числовой код (код ASCII). О нем речь пойдет в гл. 5. Именно числовые коды и сравниваются в действительности в MSX-Бейсике. Если сравнить коды каждой буквы в обеих строках, то мы увидим:

"ALGY"	65	76	71	88
"algy"	97	108	103	121

Сначала сравниваются первые символы строк, а в случае их равенства – вторые и последующие (так, ALGY меньше, чем ALGy).

ЦИКЛЫ

Условные предложения можно использовать для организации циклов.

Циклы с условием продолжения (while loops). Один из способов организации циклической структуры основан на использовании операторов IF...THEN и GOTO. В этом виде цикла последовательность операторов будет выполняться до тех пор, пока верно некоторое условие. В программе 2.7 показан пример работы такого цикла.

Программа 2.7

```
10 REM •
20 REM • Цикл с условием продолжения
30 REM •
40 C=1
50 IF C<0 OR C>9 THEN 100
60 READ NUMBER
70 PRINT NUMBER^2
80 C=C+1
90 GOTO 50
100 END
110 DATA 1,5,12,6,23,54,34,4,8,10,20,100
```

Здесь значения из списка данных оператора DATA успешно считываются и печатаются, пока значение С больше нуля, но меньше 10. Подобную логическую структуру (как ее принято называть) можно проиллюстрировать блок-схемой, приведенной на рис. 2.2.

Важным свойством рассматриваемой структуры является расположение оператора проверки окончания цикла (в нашем примере — строка 50) перед блоком выполняемых операторов. Это исключает появление сообщения об ошибке "Out of data" (нет данных).

Циклы с условием завершения (repeat loops). Другой вид циклов, организуемых с помощью условных операторов, — циклы с условием завершения. В этом случае набор операторов выполняется, если определен-

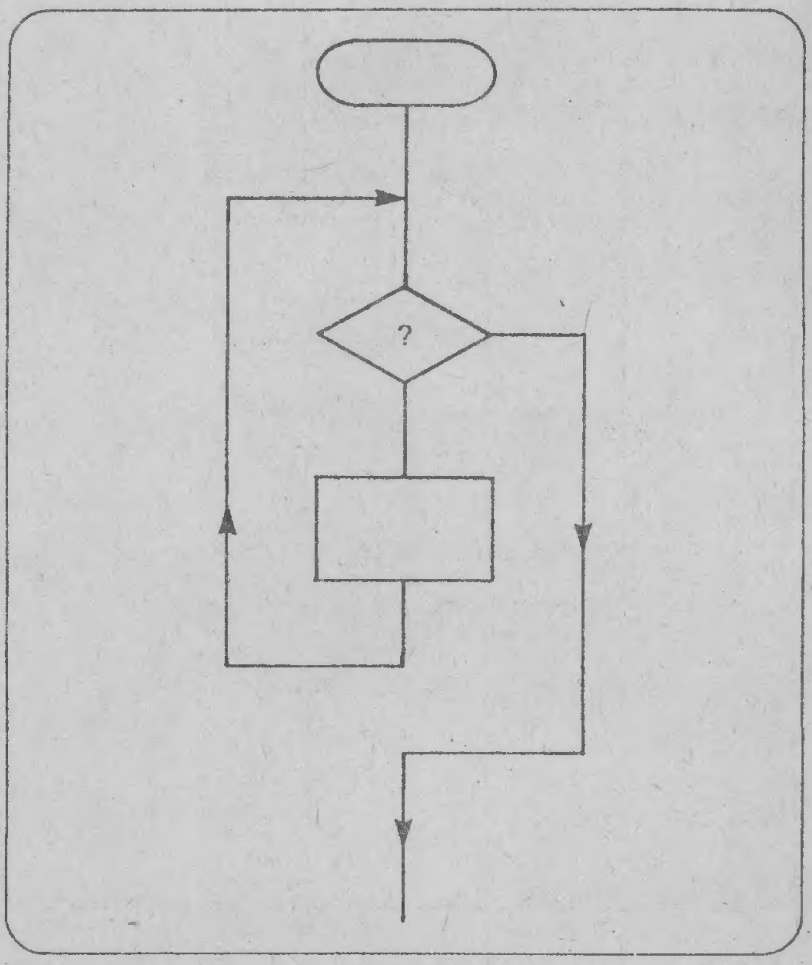

Рис. 2.2. Цикл с условием продолжения

ное условие истинно. В программе 2.8. ввод данных требуется повторять до тех пор, пока вводимая величина не окажется в диапазоне от 0 до 10. Блок-схема циклической структуры такого типа показана на рис. 2.3.

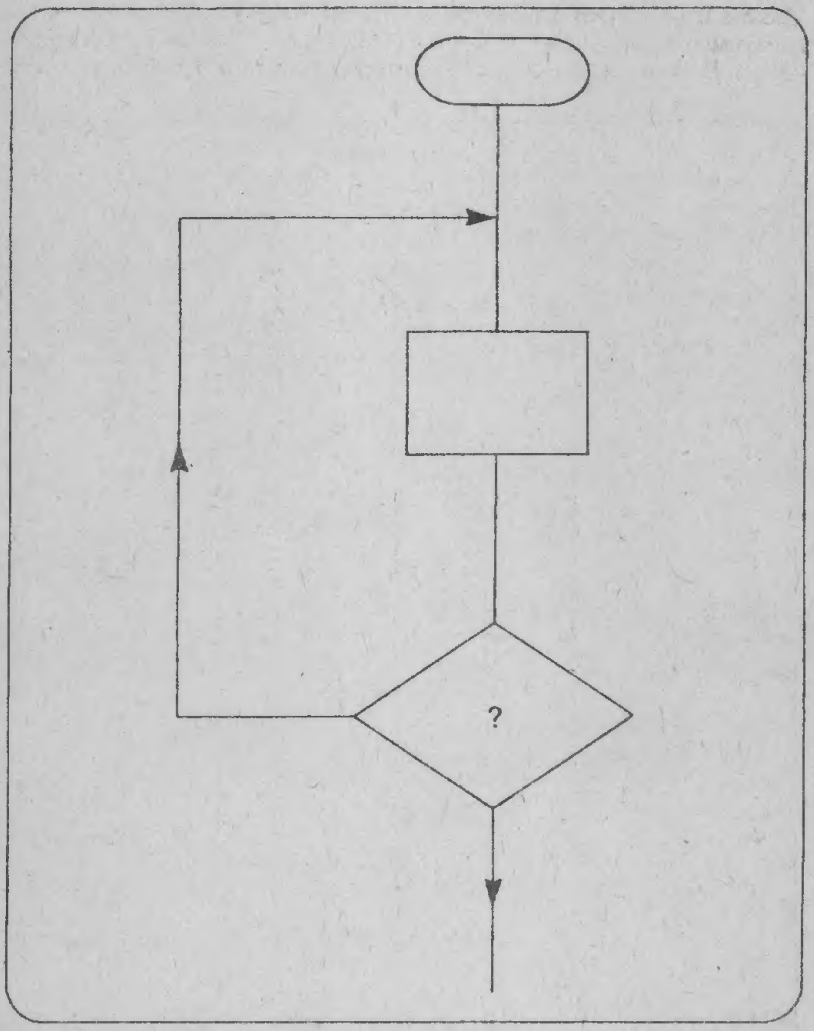

Рис. 2.3. Цикл с условием завершения

Программа 2.8

```
10 REM *
20 REM * Цикл с условием завершения
30 REM *
40 INPUT "Введите число от 0 до 10": A
50 IF A < 0 OR A > 10 THEN 40
60 PRINT "Верно"
70 END
```

Переключатели. Оператор ON...GOTO вызывает переход к одному из нескольких номеров строк в зависимости от значения некоторой переменной. Общий формат этого оператора:

ON <переменная> GOTO <номер строки>[,<номер строки>][, <номер строки>]...

Если значение переменной равно единице, выполняется переход к первому в списке номеру строки, если оно равно двум — ко второму по порядку и т.д. В тех случаях, когда значение переменной превышает количество заданных адресов перехода или когда оно меньше единицы, выполнение программы продолжается со следующей строки. Программа 2.9 вычисляет площади геометрических фигур. Задавая значение от 1 до 3, можно выбрать одну из трех фигур. Оператор ON...GOTO передаст управление соответствующей ветви программы. Возможный результат работы программы представлен на рис. 2.4.

Программа 2.9

```
10 REM *
20 REM * Вычисление площадей
30 REM * с помощью ON GOTO
40 REM *
50 CLS
60 SCREEN 0 : WIDTH 38 : KEY OFF
70 PRINT
80 PRINT "Вычисление площадей"
90 PRINT
100 PRINT "1. Прямоугольники"
110 PRINT "2. Прямоугольные треугольники"
120 PRINT "3. Круги"
130 PRINT
140 PRINT "Введите нужный номер:";
150 INPUT X
160 ON X GOTO 190,280,370
170 BEEP
180 GOTO 50
190 REM *
200 REM * Прямоугольники
210 REM *
220 CLS
230 INPUT "Введите длину стороны A":A
```

```
240 INPUT "Введите длину стороны B".B
250 PRINT
260 AREA = A * B
270 GOTO 480
280 REM *
290 REM * Треугольники
300 REM *
310 CLS
320 INPUT "Введите длину основания":B
330 INPUT "Введите высоту":H
340 PRINT
350 AREA = (B * H)/2
360 GOTO 480
370 REM *
380 REM * Круги
390 REM *
400 CLS
410 INPUT "Радиус":R
420 AREA = 3.142 * (R*R)
430 GOTO 480
440 REM *
450 REM * Печать результата
460 REM *
470 PRINT
480 PRINT "Площадь = ":AREA:" кв. ед."
490 PRINT
500 PRINT "Для другого расчета введите 1"
510 INPUT "Для окончания – любое другое число":B
520 ON B GOTO 50
530 END
```

```
Вычисление площадей
1. Прямоугольники
2. Прямоугольные треугольники
3. Круги

Введите нужный номер:? 2

Введите длину основания? 13
Введите высоту? 45

Площадь = 292.5  кв. ед.
Для другого расчета введите 1
Для окончания – любое другое число? 2
Ok
```

Рис. 2.4. Результат работы программы 2.9

ОЧИСТКА ПЕРЕМЕННЫХ

Перед тем как переменная в программе получает определенное значение, ее величина устанавливается равной нулю для числовых переменных или *пустой строке* (" ") – для строк.

Команда Бейсика CLEAR имеет многоцелевое назначение. Все числовые переменные она обнуляет, а строковые – очищает. Кроме того, ее используют для увеличения объема памяти, отведенной под текстовые данные. Напомним: вначале подразумевается, что отведенного в памяти места достаточно для 200 знаков. Если же необходимо работать с числом символов, равным 1000, нужно выполнить команду

```
CLEAR 1000
```

Еще одно назначение этой команды рассмотрено в приложении 6.

Выяснить величину свободной области памяти, отведенной под тексты, можно с помощью команды

```
FRE (" ")
```

Аналогично команда

```
FRE (0)
```

указывает размеры свободного пространства (в байтах), предназначенного для программ, переменных и данных.

Эти две команды представляют собой примеры функций, которые будут подробно рассмотрены в гл. 3.

ГЛОБАЛЬНЫЕ ТИПЫ ПЕРЕМЕННЫХ

В MSX-Бейсике предполагается, что первоначально все переменные являются вещественными, с двойной точностью. Указанное соглашение можно изменить, если явно указать другой тип переменной с помощью соответствующего символа описания типа данных. Кроме того, существуют четыре команды, которые позволяют присвоить определенный тип данных любой переменной или набору переменных. Пусть, например, все переменные, имена которых начинаются с буквы I, нужно описать как целые. Для этого достаточно ввести команду

```
DEFINT I
```

Один и тот же тип можно присвоить и переменным, имена которых начинаются с разных букв. Так, в результате выполнения команды

```
DEFINT A-Z
```

все переменные (поскольку имя любой переменной должно начинаться с латинской буквы) будут рассматриваться как целые.

Перечислим все четыре команды описания разных типов данных:

DEFINT	целочисленные переменные
DEFSNG	вещественные с одинарной точностью

DEFDBL	вещественные с двойной точностью
DEFSTR	строковые переменные

После выполнения любой из этих команд тип одиночной переменной можно вновь изменить обычным способом, с помощью символов описания типа данных.

В заключение отметим, что в настоящей главе вы получили сравнительно большой объем важной информации. Применяя рассмотренные выше команды и операторы, можно пытаться разрабатывать довольно сложные программы.

СВОДКА СЛУЖЕБНЫХ СЛОВ

CLEAR <набор символов>[,<максимальное значение адреса памяти>]
DATA <список констант>
DEF/INT/SNG/DBL/STR <ряд букв>
FRE (0)
FRE ("")
GOTO <номер строки>
IF <условное выражение> THEN <список операторов>|<номер строки> [ELSE <список операторов>|<номер строки>]
IF <условное выражение> GOTO <номер строки> [ELSE <список операторов>|<номер строки>]
INPUT [<подсказка>;]<список переменных>
LET <переменная> = <выражение>
ON <выражение> GOTO <список номеров строк>
READ <список переменных>
RESTORE [<номер строки>]
SWAP <переменная>,<переменная>

Глава 3. ФУНКЦИИ И ПОДПРОГРАММЫ

В самых разнообразных программах встречаются специальные математические выражения или наборы инструкций, которые в ходе выполнения программы должны многократно повторяться. В Бейсике есть два удобных приема для сокращения подобных записей, позволяющие записать набор инструкций или повторяющееся выражение всего лишь один раз, а использовать их в дальнейшем столько раз, сколько требуется. Это достигается с помощью двух важнейших элементов языка — функций и подпрограмм, рассмотрению которых и посвящена настоящая глава.

ВСТРОЕННЫЕ ФУНКЦИИ

С помощью функций можно вычислять простые выражения и получать в каждом случае однозначно определенный результат. В языке MSX-Бейсик имеются средства для решения таких часто встречающихся задач, как извлечение квадратных корней, нахождение синусов, косинусов и т.д. Это так называемые ***встроенные функции***. Существует более 40 таких функций, доступных программисту. ***Функции пользователя*** должны быть определены самим программистом и позволяют получить любой желаемый результат.

Прежде чем перейти к описанию встроенных математических функций, требуется ввести ряд общих терминов. Для задания функции необходимы данные, которые называются ***параметрами***, или ***аргументами***. Функции нельзя присвоить значение тем же способом, что и простой переменной. Параметр может быть переменной, константой или выражением Бейсика. Значение, полученное при вычислении функции, как говорят, "возвращается в программу". Обращение к функции Бейсика выполняется по ее имени, за которым следуют скобки, например SIN( ). Ниже описаны некоторые широко распространенные математические функции.

Тригонометрические функции. В MSX-Бейсике предусмотрена возможность работы с четырьмя тригонометрическими функциями: SIN( ), COS( ), TAN( ), ATN( ) (синус, косинус, тангенс, арктангенс). Обычно мы работаем с выражениями вида: X =SIN(45), что означает "положим X равным синусу угла в 45 градусов". В данном случае MSX-Бейсик

возвращает значение, приблизительно равное 0.8509, что заметно отличается от ожидаемой величины (0.7071). Происходит это потому, что тригонометрические функции вычисляются от аргумента в радианах, а не в градусах. В полной окружности содержится 2*PI радиан, где число "пи" приближенно равно 3.142. Таким образом, один градус равен 2*PI/360 радиан.

Все тригонометрические функции можно определить с помощью рис. 3.1. Здесь

```
синус X     =   B / H
косинус X   =   A / H
тангенс X   =   (B / H) / (A / H)
```

Арктангенс ATN( ) является обратной функцией по отношению к TAN( ). В программе 3.1 введенные обозначения используются для печати значений всех тригонометрических функций, причем угол можно задавать как в радианах, так и в градусах. Типичный вариант результата приведен на рис. 3.2.

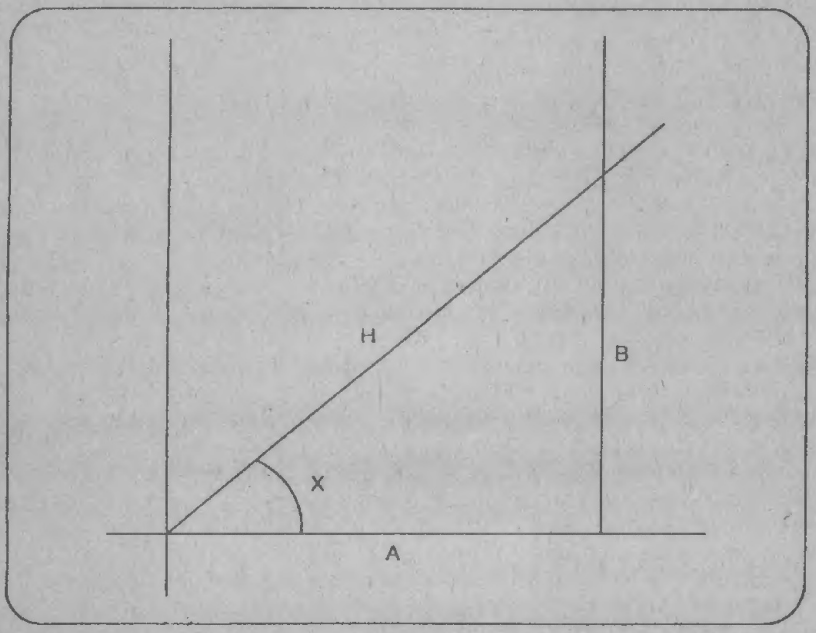

Рис. 3.1. К определению тригонометрических функций

Программа 3.1

```
10 REM *
20 REM * Тригонометрические функции
30 REM *
40 SCREEN 0 : KEY OFF
50 PRINT "Градусы (1) или радианы (2) ";
60 INPUT SWITCH%
70 IF SWITCH% < 1 OR SWITCH% > 2 THEN
GOTO 50
80 PRINT
90 PRINT "Введите угол";
100 INPUT A
110 IF SWITCH% = 2 THEN GOTO 160
120 REM *
130 REM * Перевод в радианы
140 REM *
150 A = A * 3.14159/180
160 REM *
170 REM * Печать значений функций
180 REM *
190 PRINT
200 PRINT "Синус угла     :";SIN(A)
210 PRINT "Косинус угла  :";COS(A)
220 PRINT "Тангенс угла  :";TAN(A)
230 PRINT "Арктангенс угла:";ATN(A)
240 PRINT
250 PRINT "Угол =";A;"радиан."
```

```
Градусы (1)          или          радианы (2)  1

Введи е угол         36

Синус угла           :            .58778482293256
Косинус угла         :            .809017306323
Тангенс угла         :            .72654171714082
Арктангенс угла      :            .5609817356067

Угол = .628318 радиан.
Ok
```

Рис. 3.2. Значения тригонометрических функций из программы 3.1

INT( ) и FIX( ). Эти функции используются для округления вещественных чисел до целых и могут применяться в разнообразных ситуациях. Обе они возвращают значение целой части вещественного числа, но

различаются по способу округления. Функция INT( ) округляет число "вниз" до ближайшего целого числа. FIX( ) действует в зависимости от знака числа: отрицательные числа округляются "вверх" до ближайшего к ним целого, а положительные – "вниз". Программа 3.2 помогает лучше уяснить различие между этими двумя функциями.

Программа 3.2

```
10 REM *
20 REM * Функции INT( ) и FIX( )
30 REM *
40 SCREEN 0 : KEY OFF
50 C=5
60 READ A
70 PRINT "Начальное значение =";A
80 PRINT "FIX( )      =";FIX(A)
90 PRINT "INT( )      =";INT(A)
100 PRINT
110 C=C-1
120 IF C<>0 THEN GOTO 60
130 DATA 12.453
140 DATA -1932.555
150 DATA -3.21
160 DATA 234.11
170 DATA -0.55
```

Преобразования типов данных. Типы данных числовых величин можно изменять. При этом меняются размеры памяти, отведенной для хранения чисел, а также может происходить округление. Ниже рассматриваются основные функции преобразования числовых величин.

CINT( ). Преобразует вещественное число в целое путем отсечения его дробной части. В отличие от FIX( ) или INT( ), CINT( ) оперирует только с вещественными числами в диапазоне, разрешенном для целых чисел.

CSNG( ). Преобразует целые и вещественные числа в вещественные одинарной точности. Под число в памяти отводится четыре байта. Величины с двойной точностью при таком преобразовании округляются вверх или вниз до шести значащих цифр.

CDBL( ). Преобразует числа в числа двойной точности. Размер отводимой для числа памяти увеличивается до восьми байтов.

BIN$( ). Преобразует формат целого числа в двоичный символьный формат, не изменяя тип аргумента.

HEX$( ). Как и функция BIN$( ), возвращает строку символов, но при этом целое число преобразуется в шестнадцатиричные коды.

OCT$( ). Функция действует в тех же условиях, что и HEX$( ) и BIN$( ), возвращая целое число в восьмиричном формате.

В программе 3.3 приводятся примеры преобразований с помощью перечисленных выше функций.

Программа 3.3

```
10 REM *
20 REM * Функции преобразования типов
30 REM *
40 SCREEN 0 : KEY OFF
50 C=2
60 READ A
70 PRINT "Исходное число = ";A
80 PRINT "CINT =";CINT(A)
90 PRINT "CSNG =";CSNG(A)
100 PRINT "CDBL =";CDBL(A)
110 C = C-1
120 IF C <>0 THEN GOTO 60
130 PRINT
140 C=3
150 READ A
160 PRINT "Исходное число = ";A
170 PRINT "BIN$ = ";BIN$(A)
180 PRINT "HEX$ = ";HEX$(A)
190 PRINT "OCT$ = ";OCT$(A)
200 C = C - 1
210 IF C <>0 THEN GOTO 150
220 DATA 0.434235
230 DATA 152.32331221231
240 DATA 1000
250 DATA 231
260 DATA -12
```

SGN( ) и ABS( ). Функция SGN( ) определяет знак числа. Она возвращает одно из трех значений:

0,	если аргумент = 0;
−1,	если аргумент < 0;
1,	если аргумент > 0.

ABS( ) возвращает абсолютное значение выражения. Эту функцию можно использовать для преобразования отрицательных чисел в положительные или для вычисления модуля разности двух чисел, например:

```
ABS( - 5.87) = 5.87
ABS(1223 - 1240) = 17
```

Случайные числа. В ряде ситуаций бывает полезным, чтобы программа содержала элемент случайности. Одно из основных применений случайных чисел связано с моделированием событий реального мира. Например, моделируя появление покупателей в магазине, можно прогнозировать необходимую численность персонала в определенное время суток. Другая область использования таких величин — игры, связанные, к примеру, с налетом бомбардировщиков или высадкой вражеского десанта; карточные игры.

Функция RND( ) генерирует последовательность (псевдо)случайных величин, зависящих от введенного аргумента. Влияние аргумента сводится к следующему:

* если аргумент равен нулю, выдается последнее из созданных случайных чисел;
* если аргумент положителен, выдается следующее число из последовательности, заданной последней инициализацией генератора;
* если аргумент отрицателен, генератор случайных чисел запускается заново с новым исходным значением, и создается новая случайная последовательность.

Программы, использующие положительные аргументы, при каждом запуске формируют одну и ту же последовательность чисел. Чтобы получить абсолютно непредсказуемые случайные числа, нужно ***случайным же образом*** задавать для функции RND( ) отрицательный аргумент. Это можно сделать двумя основными способами. Первый из них основан на введении пользователем отрицательного числа. Поскольку заранее неизвестно, какие числа будут введены в ответ на запрос оператора INPUT при последующих запусках программы, набор случайных чисел становится менее предсказуемым. Второй способ предполагает применение специальной переменной MSX-Бейсика.

Процессор 50 раз в секунду генерирует сигнал, который воспринимается MSX-Бейсиком. Текущее значение счетчика времени хранится в переменной TIME. В любой момент эта переменная имеет какое-то значение от 0 до 65535. Поскольку время запуска программы зависит лишь от желания пользователя, значение TIME нельзя предсказать заранее, что делает эту переменную идеально подходящей для случайного сброса генератора.

Возвращаемое функцией RND( ) число всегда находится в диапазоне от 0 до 1. Чтобы получить случайное число из другой шкалы, скажем от 0 до 100, нужно значение RND( ) умножить на 100.

Можно применять RND( ) для моделирования бросания двух игральных костей. Каждый раз должен получаться результат от 1 до 6, что достигается умножением значения RND( ) на 6, округлением полученного числа вниз с помощью функции INT( ) и прибавлением к результату 1. Эта маленькая хитрость позволяет избежать нежелательных нулевых значений и преобразовать моделируемую величину к заданным пределам.

Программа 3.4

```
10 REM *
20 REM * Модель двух игральных костей
30 REM *
40 REM * Сброс датчика RND( )
50 REM *
60 CLS
70 X = RND(-TIME)
80 D1 = INT(RND(1)*6)+1
90 D2 = INT(RND(1)*6)+1
```

```
100 PRINT "Кубик 1 ";D1;"Кубик 2 ";D2
110 PRINT
120 PRINT "Для продолжения нажмите RETURN"
130 INPUT X
140 PRINT
150 GOTO 80
```

В гл. 4 рассматриваются возможные варианты использования переменной TIME.

ФУНКЦИИ ПОЛЬЗОВАТЕЛЯ

В MSX-Бейсике функции могут иметь до девяти определяемых параметров. Функции пользователя всегда можно отличить от встроенных функций, так как их имена начинаются буквами FN. Рассмотрим простой пример. Пусть нужно записать функцию, вычисляющую длину окружности. Требуемое соотношение на Бейсике записывается следующим образом:

```
C = 2 * PI * R
```

где C – длина окружности, а R – ее радиус. Используя приближенное значение PI, можно определить функцию так:

```
DEF FNCIRC(R) = 2 * 3.141593 * R
```

Этой функции при создании присвоено имя FNCIRC. В скобках указан параметр, заменяемый при выполнении программы числовым значением аргумента. Все символы внутри скобок определяют локальные переменные, которые должны применяться в выражении. Переменная R в основной программе не повлияет на значение R в функции FNCIRC.

Между параметрами в скобках и их значениями в процессе использования функции существует взаимно однозначное соответствие. Если функция, определенная как

```
FNA(A,B,C) = A * B * C
```

вызывается с помощью выражения X = FNA(1,2,4), локальные переменные приобретают значения: A = 1, B = 2 и C = 4. Программа 3.5 иллюстрирует выполнение функции FNCIRC, а результаты представлены на рис. 3.3.

Программа 3.5

```
10 REM *
20 REM * Вычисление длины окружности
30 REM *
40 DEF FNCIRC(R) = 2 * 3.141593# * R
50 CLS
60 INPUT "Радиус ";R
70 C=FNCIRC(R)
80 PRINT "Длина окружности = ";C
90 END
```

```
Радиус ? 145
Длина окружности =  911.06197
```

Рис.3.3. Результат выполнения программы 3.5

Более сложный пример работы с функциями показан в программе 3.6. Две используемые в ней функции преобразуют координаты точки при повороте на определенное число градусов. Предположим, что точка с координатами (0,1) должна повернуться относительно начала координат (0,0) на 45° . Новые координаты точки будут (0.707,0.707). Аналогично, та же точка при повороте на 90° получит координаты (1,0). От пользователя требуется ввести начальные координаты точки X и Y и величину угла поворота в градусах. Затем функция пользователя переводит градусы в радианы. Две другие функции вычисляют результат вращения точки (X,Y). Мы вернемся к этому примеру в гл. 8 при рассмотрении программ машинной графики.

Программа 3.6

```
10 REM *
20 REM * Вращение точки
30 REM *
40 DEF FNC(D)=D*3.141599#/180
50 DEF FNXC(X,Y,R)=X*COS(R)+Y*SIN(R)
60 DEF FNYC(X,Y,R)=-(X*SIN(R)-Y*COS(R))
70 SCREEN 0 : KEY OFF
80 INPUT "Координаты точки ":X,Y
90 INPUT "Угол поворота ":A
100 PRINT
110 R=FNC(A)
120 NX=FNXC(X,Y,R)
130 NY=FNYC(X,Y,R)
140 PRINT "X =":NX
150 PRINT "Y =":NY
160 END
```

Следует отметить, что выражение, определяющее функцию, может содержать другие функции пользователя. Например, последовательность операторов

```
DEF FNA(A) = A / 100 * 3
DEF FNB(B) = B ^ (1/2)
DEF FNC(X,Y,Z) = FNA(X) + FNB(Y) * Z
```

не выполняющая сколько-нибудь содержательных действий, показывает, что такое включение ранее определенных функций в MSX-Бейсике вполне возможно.

ПОДПРОГРАММЫ

Применение функций дает хорошие результаты, если требуется вычислить значение только одной величины. Для решения же серьезных задач, когда требуется более сложная последовательность операторов, не сводящаяся к одной функции.

Точно так же, как применение функции избавляет от необходимости многократно переписывать одиночное выражение, подпрограмма позволяет записывать повторяющуюся группу операторов всего один раз. Использование подпрограмм не только освобождает программиста от нудного переписывания фрагментов текста, но и улучшает структуру программы. Для каждого из логических этапов решения задачи можно написать подпрограмму.

Подпрограммы не объявляются в явном виде, но к ним можно обращаться. Обращение осуществляется в форме команды GOSUB. Она выполняется сходным образом с командой GOTO, но между ними имеется одно существенное различие. Команда GOTO вызывает переход к заданному номеру строки, и в дальнейшем программа будет выполняться с указанного места до конца. При ветвлении по команде GOSUB происходит переход к заданному номеру строки, и программа выполняется до тех пор, пока не встретится оператор RETURN, после чего осуществляется возврат к строке, непосредственно следующей за командой GOSUB. Этот процесс иллюстрируется программой 3.7.

Программа 3.7

```
10 REM *
20 REM * Демонстрация работы GOSUB
30 REM *
40 SCREEN 0 : KEY OFF
50 GOSUB 130
60 PRINT "Снова на строке 60"
70 GOSUB 150
80 PRINT "И снова на строке 80"
90 END
100 REM *
110 REM * Подпрограммы
120 REM *
130 PRINT "Привет от строки 130"
140 RETURN
150 PRINT "Привет от строки 150"
160 PRINT "и 160"
170 PRINT "и 170"
180 PRINT "и 180"
190 RETURN
```

В подпрограмме допускается наличие нескольких операторов RETURN. Они позволяют возвращаться в основную программу в зависимости от выполнения определенных условий.

Можно также пользоваться оператором RETURN для передачи управления любой другой строке Бейсик-программы, например RETURN 50. Эта особенность языка не относится к числу наиболее полезных и без нее легко можно обойтись. Возможно, вместо применения оператора RETURN с номером строки лучше имитировать подпрограмму с помощью двух операторов GOTO.

Допускается также гнездование подпрограмм, т.е. одна подпрограмма может обращаться к другой. Еще одно интересное свойство подпрограмм называется рекурсивностью: подпрограмма может содержать обращение к себе самой. Рассмотрим такую задачу. Пишется подпрограмма для проверки данных, вводимых в программу. Если очередное значение не попадает в разрешенный диапазон, выдается сигнал об ошибке и запрашивается следующее число. Приведем алгоритм этой подпрограммы (пусть она называется VALID):

1. Запросить возраст пользователя.

2. Если значение возраста < 16 или > 65, подать звуковой сигнал (BEEP) и перейти к п.1 (GOSUB VALID).

3. В противном случае, если данные являются допустимыми, вернуться в подпрограмму (RETURN).

Этот особый случай работы с подпрограммами может служить мощным средством программирования, если им аккуратно пользоваться.

МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

Анализируя сложную задачу, можно последовательно разбить ее на ряд более простых, для которых затем разрабатывается программная реализация. Такой процесс называется ***модульным программированием***, или ***программированием сверху вниз***.

В качестве примера проанализируем задачу составления платежной ведомости. Для определенного числа служащих требуется ввести следующие данные:

1. Имя служащего.

2. Число часов, отработанных за прошедшую неделю.

3. Размер почасовой тарифной ставки.

Программа должна рассчитать недельную зарплату каждого из них с учетом приведенной ниже информации:

1. Ни один работник не имеет права работать свыше 50 часов в неделю.

2. Недельная зарплата более 40 фунтов облагается 30%-ным налогом.

3. При работе более 37 часов в неделю сверхурочное время оплачивается в размере 150% часовой ставки.

Для решения задачи можно предложить такую последовательность действий:

1. Запросить число служащих, для которых выполняется расчет.

2. Ввести данные для очередного служащего.

3. Вычислить количество сверхурочных часов (если это необходимо).
4. Вычислить размер сверхурочной оплаты (если это необходимо).
5. Вычислить размер зарплаты за обычные часы работы.
6. Вычислить полную величину зарплаты.
7. Вычислить размер налога (если это необходимо).
8. Напечатать всю нужную информацию.
9. Если обработаны данные не для всех служащих, повторить действия 2–8.

Теперь все существенно упрощается. Ясно, что часть перечисленных действий можно реализовать в виде подпрограмм. Вместо того чтобы пытаться решать задачу целиком, возьмем небольшой ее фрагмент и разберем его досконально. Ясно, что количество отработанных часов не может быть ни отрицательным, ни большим 50. Величина почасовой ставки также заранее не известна, но она никак не может составлять меньше 0.00 фунтов в час. Итак, алгоритм процедуры ввода данных:

* Ввести имя сотрудника.
* Ввести количество отработанных часов.
* Если это количество < 0 или > 50, дать сообщение об ошибке и вновь запросить данные.
* Ввести величину почасовой ставки.
* Если эта величина < 0, дать сообщение об ошибке и вновь запросить данные.
* Завершить выполнение подпрограммы.

Поэтапное программирование упрощает составление и отладку программы 3.8, представляющей собой полный текст реализованной на Бейсике задачи составления платежной ведомости. Большая часть программы имеет модульную структуру, что позволяет при необходимости легко вносить исправления. Подпрограммы могут в дальнейшем включаться в новые программы.

Программа 3.8

```
10 REM ************************
20 REM *                      *
30 REM * Программа "Зарплата" *
40 REM *                      *
50 REM ************************
60 REM *
70 REM * Основная программа
80 REM *
90 SCREEN 0 : KEY OFF
100 WIDTH 38
110 PRINT "Введите численность"
120 PRINT "занятых работников."
130 INPUT ":";N%
140 IF N% <0 THEN BEEP : GOTO 130
150 REM *
160 REM * Главный цикл
170 REM *
```

```
180 GOSUB 320 : REM • Ввод данных
190 GOSUB 420 : REM • Расчет сверхурочных
200 GOSUB 500 : REM • Начисление налога
210 GOSUB 590 : REM • Подпрограмма печати
220 N% = N%+1
230 REM •
240 REM • После расчета для всех сотрудников – конец
250 REM •
260 IF N%=0 THEN PRINT "Конец": END
270 INPUT "Для следующего расчета нажмите Return";A$
280 GOTO 180
290 REM •
300 REM • Подпрограмма ввода
310 REM •
320 CLS
330 INPUT "Имя       : ";N$
340 INPUT "Отработанные часы ";HW
350 IF (HW<0) OR (HW>50) THEN BEEP :GOTO 340
360 INPUT "Почасовая ставка : ";RP
370 IF (RP<0) THEN BEEP : GOTO 360
380 RETURN
390 REM •
400 REM • Расчет сверхурочных часов
410 REM •
420 IF (HW<=37) THEN OH=0:OP=0 : RETURN
430 OH = HW-37
440 HW = 37
450 OP = (1.5 • RP) • OH
460 RETURN
470 REM •
480 REM • Начисление зарплаты и налога
490 REM •
500 BP = (HW • RP)
510 GP = BP + OP
520 IF GP<=40 THEN TAX=0:NP=GP:RETURN
530 TAX = GP*.3
540 NP = GP - TAX
550 RETURN
560 REM •
570 REM • Вывод подробной информации
580 REM •
590 CLS
600 PRINT "Полный итог"
610 PRINT
620 PRINT "Имя       : ";N$
630 PRINT
640 PRINT "Отработанные часы :";HW
650 PRINT "Почасовая ставка  :";RP
660 PRINT
```

```
670 PRINT "Основная зарплата :";BP
680 PRINT "Сверхурочные часы :";OH
690 PRINT "Оплата сверхурочных:";OP
700 PRINT
710 PRINT "Суммарная зарплата :";GP
720 PRINT "Сумма налогов:";TAX
730 PRINT
740 PRINT "К выплате     :";NP
750 PRINT
760 RETURN
```

СОЗДАНИЕ БИБЛИОТЕК ПОДПРОГРАММ

Подпрограммы, используемые в нескольких программах, можно хранить на ленте и вводить при необходимости. Для записи подпрограмм лучше применять команду SAVE, а не CSAVE. Если нужно, их можно присоединять к программе по команде MERGE. Однако следует отметить, что если основная программа содержит строки с такими же номерами, что и подпрограмма, присоединяемая по команде MERGE, то эти строки будут заменены соответствующими строками подпрограммы.

СВОДКА СЛУЖЕБНЫХ СЛОВ

ABS(X), ATN(X), BIN$(X), CDBL(X), CINT(X), COS(X), CSNG(X), FIX(X), HEX$(X), INT(X), OCT$(X), RND(X), SGN(X), SIN(X), TAN(X)
DEF FN<имя>[(<список параметров>)]=<определение функции>
GOSUB <номер строки>
RETURN [<номер строки>]
MERGE <имя файла>

Глава 4. ЦИКЛЫ, ПРЕРЫВАНИЯ И УПРАВЛЕНИЕ ПРОЦЕССАМИ

В системе имеется несколько устройств ввода, которыми можно управлять с помощью специальных команд Бейсика. В настоящей главе вводятся некоторые из этих команд, но прежде чем перейти к их рассмотрению, поговорим еще немного о циклах.

ЦИКЛ FOR...NEXT

В гл. 2 было показано, как организуются программные циклы с применением условных операторов. В MSX-Бейсике предусмотрен более удобный способ создания повторяющихся циклов. Он основан на использовании двух операторов:

FOR <счетчик цикла> = <начальное значение> TO <конечное значение> [STEP <приращение>]

и

NEXT [<переменная>][,<переменная>][,<переменная>]...

Работу с такой конструкцией лучше всего показать на примере. Предположим, требуется напечатать максимальное и минимальное значения для набора из 10 чисел, хранящихся в операторе DATA. Решим эту задачу двумя способами: сначала с помощью операторов IF...THEN, образующих повторяющийся цикл (программа 4.1), а затем с помощью эквивалентно действующих операторов FOR...NEXT (программа 4.2).

Программа 4.1

```
10 REM *
20 REM * Поиск максимума и минимума
30 REM * с помощью цикла IF...THEN
40 REM *
50 CLS
60 MX=0 : MN=10^61
70 I = 1
80 REM * Главный цикл
90 READ N
```

```
100 PRINT I:"",N
110 IF N>MX THEN MX=N
120 IF N<MN THEN MN=N
130 I = I+1
140 IF I<=10 THEN 90
150 REM *
160 REM * Печать результата
170 REM *
180 PRINT
190 PRINT "Максимальное значение = ";MX
200 PRINT "Минимальное значение = ";MN
210 END
220 DATA 675,43,-4.24,49241,2
230 DATA 12,-1032,999,89.65,34
```

Программа 4.2

```
10 REM *
20 REM * Поиск максимума и минимума
30 REM * с помощью цикла FOR...NEXT
40 REM *
50 CLS
60 MX=0 : MN=10^61
70 REM * Главный цикл
80 FOR I=1 TO 10
90 READ N
100 PRINT I:"",N
110 IF N>MX THEN MX=N
120 IF N<MN THEN MN=N
130 NEXT I
140 REM *
150 REM * Печать результата
160 REM *
170 PRINT
180 PRINT "Максимальное значение = ";MX
190 PRINT "Минимальное значение = ";MN
200 END
210 DATA 675,43,-4.24,49241,2
220 DATA 12,-1032,999,89.65,34
```

В обеих программах переменная I служит счетчиком числа выполненных шагов. В программе 4.2 оператор

```
NEXT I
```

эквивалентен оператору

```
I = I + 1
```

из программы 4.1. Оператором FOR устанавливается начальное значение I, а также проверяется, не превысило ли в ходе выполнения программы

текущее значение I предельную величину <конечное значение>, в данном случае 10.

Такая запись оператора цикла значительно компактнее и гораздо легче воспринимается при чтении, чем конструкция цикла с оператором IF...THEN. Имя переменной в операторе NEXT разрешается опускать, но, как правило, его лучше сохранить для большей ясности.

В программе 4.2 оператор цикла FOR действует как простой счетчик. Переменную цикла можно использовать и внутри самого цикла.

Факториал числа определяется следующим выражением:

$$n! = 1 * 2 * 3 \ldots * n$$

(читается: n – факториал). Например:

$$\begin{aligned} 3! &= 1 * 2 * 3 \\ &= 6 \\ 8! &= 1 * 2 * 3 * 4 \ldots * 8 \\ &= 40320 \end{aligned}$$

Цикл FOR...NEXT позволяет вычислить факториал любого числа. Для этого запоминается промежуточное значение результата, которое при каждом прохождении цикла умножается на текущее значение I (см. программу 4.3 и рис. 4.1). В результате выполнения программы быстро достигается значение, максимально допустимое в MSX-Бейсике, поэтому выбор в качестве N слишком большого числа приводит к появлению сообщения об ошибке "Overflow" (переполнение).

Программа 4.3

```
10 REM *
20 REM * Факториалы
30 REM *
40 CLS
50 SUM = 1
60 INPUT "Число ";N
70 FOR I = 1 TO N
80 SUM = SUM * I
90 NEXT
100 PRINT
110 PRINT N;"! = ";SUM
120 END
```

```
Число ? 12

12 ! = 479001600
```

Рис. 4.1. Результат выполнения программы 4.3

Использование параметра STEP. Когда интерпретатор обрабатывает оператор NEXT, обычно значение параметра цикла увеличивается на единицу. С помощью параметра STEP в операторе FOR можно задать увеличение (или уменьшение) значения счетчика на любое число. В программе 4.4 параметр STEP используется для вывода в радианах всех углов от 0 до 360° с шагом 45°. Результат представлен на рис. 4.2.

Программа 4.4

```
10 REM *
20 REM * Параметр STEP
30 REM *
40 A=0
50 PRINT "Градусы","Радианы"
60 PRINT
70 FOR I = 0 TO 360 STEP 45
80 R=I*(3.142/180)
90 PRINT I,R
100 NEXT
```

Градусы	Радианы
0	0
45	.785500000000002
90	1.571
135	2.3565000000001
180	3.1420000000001
225	3.9275000000001
270	4.7130000000001
315	5.4985000000001
360	6.2840000000002

Рис. 4.2. Перевод градусов в радианы с помощью программы 4.4

Разумеется, отрицательное значение STEP вызывает уменьшение величины счетчика цикла. Как и в цикле повторяющейся структуры, все операторы, заключенные между FOR и NEXT, будут *всегда* выполняться по меньшей мере один раз. Даже такой невероятный оператор, как

```
FOR I = 12 TO 15 STEP -1
```

вызовет однократное выполнение цикла, несмотря на то, что конечное значение параметра никогда не достигается.

Циклы такого типа применяются для организации временных задержек при исполнении программ и называются ***пустыми циклами***, поскольку их единственное назначение – тянуть время. Ниже приведен пример пустого цикла. Длительность вызванной им паузы определяется, помимо числа повторений, типом и точностью переменной цикла.

```
FOR D = 1 TO 1000 : NEXT D
```

Вложенные циклы. Внутри одного цикла FOR...NEXT может содержаться другой цикл. Такая ситуация называется ***вложением циклов***, или ***гнездованием***. Вложенные циклы организуются следующим образом:

```
10 FOR I = 1 TO 10
20 FOR J = 1 TO 10
       (операторы программы)
50 NEXT J
60 NEXT I
```

Оба оператора FOR и NEXT внутреннего цикла должны находиться строго между FOR и NEXT внешнего цикла. За этим нужно следить, поскольку перекрещивание циклов недопустимо. Приведем пример неверной организации вложенных циклов:

```
10 FOR I = 1 TO 10
20 FOR J = 1 TO 10
       (операторы программы)
50 NEXT I
60 NEXT J
```

При гнездовании достаточно лишь одного оператора NEXT. Так,

```
NEXT K,J,I
```

завершает сразу три цикла. Однако такое обозначение не столь понятно, как набор из трех операторов NEXT, что может неоправданно усложнить исправление ошибок в программе.

Простой способ сделать вложенные циклы более наглядными при чтении программы – расположить их с парными отступами:

```
10  FOR I = 1 TO 10
20        FOR J = 1 TO 10
30              FOR K = 1 TO 10
                  (операторы программы)
70              NEXT K
80        NEXT J
90 NEXT I
```

При этом соответствующие друг другу операторы FOR и NEXT видны намного отчетливее, однако наличие дополнительных пробелов увеличивает объем памяти, необходимой для хранения программы.

Вложенные циклы использованы в программе 4.5 для печати песенки "One Man Went To Mow (Один человек пошел к стогу)". Компьютер "исполняет" песенку вплоть до куплета о том, как "10 человек пошли к стогу". Пример выглядит несколько искусственным, но тем не менее он хорошо иллюстрирует работу с вложенными циклами (см.рис. 4.3).

Программа 4.5

```
10 REM •
20 REM • Печать слов песенки
30 REM •
40 CLS
50 A$ = " One man and his dog
60 B$ = " went to mow a meadow"
70 PRINT " One man went to mow
80 PRINT " went to mow a meadow"
90 PRINT A$ : PRINT B$
100 FOR I=2 TO 10
110 REM • Цикл временной задержки
120 FOR D = 1 TO 800:NEXT D
130 CLS
140 BEEP
150 PRINT I;" men went to mow
160 PRINT " went to mow a meadow"
170 FOR J=I TO 2 STEP -1
180 PRINT J;" men"
190 NEXT J
200 PRINT A$ : PRINT B$
210 NEXT
220 PRINT "Конец"
230 END
```

```
10  men went to mow
went to mow a meadow
10  men
9  men
8  men
7  men
6  men
5  men
4  men
3  men
2  men
One man and his dog
went to mow a meadow

Конец
```

Рис. 4.3. Последний куплет песенки, получаемой по программе 4.5.

УПРАВЛЕНИЕ УСТРОЙСТВАМИ

Все компьютеры MSX имеют как минимум один порт ввода-вывода. Как правило, этот порт используется для подключения координатной рукоятки (джойстика), но позволяет работать и с другими, реже применяемыми устройствами. В MSX-Бейсике предусмотрен ряд функций для работы с такими устройствами. Важнейшей из них, несомненно, является функция STICK( ).

В качестве значения параметра функции STICK( ) можно брать одно из трех чисел:

0	чтение символа с клавиатуры;
1	чтение символа с джойстика, подключенного к порту A;
2	чтение символа с джойстика, подключенного к порту B.

STICK( ) возвращает значение, определяемое текущим положением данного джойстика. Приведем обозначения для направлений перемещения курсора:

0	нейтральное (в центре)	5	вниз
1	вверх	6	вниз и влево
2	вверх и вправо	7	влево
3	вправо	8	вверх и влево
4	вниз и вправо		

Все программы, приведенные в книге, используют STICK(0), таким образом, роль джойстика играют клавиши управления курсором, расположенные на клавиатуре. Эта функция детально описывается в последующих главах, поэтому здесь мы не будем ее подробно рассматривать.

Обычно у джойстика есть одна пусковая кнопка (у некоторых моделей – две). За их состоянием можно следить с помощью команд Бейсика. Функция STRIG( ) определяет, нажата или нет данная кнопка. Эта функция допускает следующие параметры:

0	клавиша пробела (если в качестве "встроенного" джойстика используется клавиатура);
1, 3	пусковые кнопки джойстика, подключенного к порту A;
2, 4	пусковые кнопки джойстика, подключенного к порту B.

Функция STRIG( ) возвращает значение –1 (при нажатой кнопке) либо 0 (если кнопка не нажата). Эти два значения являются примерами логических (булевских) величин в системе MSX. "Истина" соответствует числу –1, в то время как 0 или любое другое число означают "ложь". Следующее выражение, возможно, выглядит несколько странно, но имеет вполне определенный смысл:

```
IF STRIG(0) THEN BEEP
```

Оно вызывает звуковой сигнал (BEEP), если кнопка джойстика нажата (т.е. STRIG(0) имеет значение –1).

При использовании этой функции в процессе игры, когда требуется, например, вести "стрельбу", функция STRIG( ) ежесекундно задействуется по многу раз. Для таких частых проверок значений существует термин "опрос".

Метод опроса иллюстрируется программой 4.6. Программа начинается с вопроса пользователю о том, располагает ли он джойстиком или работает за клавиатурой. Далее запрашиваются состояния пусковых кнопок, и в зависимости от того, какая из них нажата, выбирается джойстик.

Программа 4.6

```
10 REM *
20 REM * Функция STRIG( )
30 REM *
40 CLS
50 WIDTH 40
60 PRINT "Нажмите пусковую кнопку "
70 PRINT "выбранного джойстика или"
80 PRINT "клавишу ПРОБЕЛ – для клавиатуры"
90 PRINT : PRINT
100 PRINT "<НАЖИМАЙТЕ>"
110 PRINT : PRINT
120 REM *
130 REM * Опрос состояния всех пусковых кнопок
140 REM *
150 F=-1
160 FOR I=0 TO 4
170 IF STRIG(I) THEN F=I
180 NEXT I
190 IF F THEN GOTO 160
200 BEEP
210 REM *
220 REM * Печать включенного устройства
230 REM *
240 IF F=0 THEN PRINT "Клавиатура";
250 IF F=1 OR F=3 THEN PRINT "Джойстик A";
260 IF F=2 OR F=4 THEN PRINT "Джойстик B";
270 PRINT "в действии":END
```

К портам, предназначенным для джойстиков, можно подключать и другие устройства, к которым также можно обращаться по командам Бейсика, например графические планшеты и игровые рычажки. Они употребляются реже, и здесь мы не приводим примеров работы с ними. Тем не менее в приложении 1 для полноты описания соответствующие функции рассматриваются.

КОНТРОЛЬ СОСТОЯНИЯ КЛАВИАТУРЫ

В MSX-Бейсике имеется возможность проверять состояние клавиатуры — с помощью функции INKEY$. Эта функция не нуждается в параметрах, ведь клавиатура у компьютера всего одна. INKEY$ возвращает значение, соответствующее нажатой клавише. Если никакой символ не вводится, INKEY$ возвращает пустую строку (" ").

Функция INKEY$ удобна в самых различных ситуациях. Один из примеров ее выполнения иллюстрируется программой 4.7.

Программа 4.7

```
10 REM *
20 REM * Функция INKEY$
30 REM *
40 CLS
50 PRINT "Нажмите любую клавишу.."
60 PRINT
70 A$ = INKEY$
80 IF A$="" THEN 70
90 BEEP
100 PRINT "Вы нажали клавишу ' ":A$;" ' "
110 END
```

Другие функции, связанные с клавиатурой, обсуждаются в гл. 5.

ОБРАБОТКА ОШИБОК

В процессе программирования могут возникнуть ошибки следующих четырех типов:

* логические, встречающиеся в том случае, когда программа нормально выполняется, но не приводит к ожидаемому результату;
* синтаксические (появляющиеся, как правило, на стадии разработки программы), в результате которых интерпретатор не может расшифровать написанный оператор;
* ошибки в работе оборудования, встречающиеся довольно редко. Последствия выхода из строя компьютера и других связанных с ним устройств невозможно предотвратить программным путем;
* ошибки в ходе выполнения программы. Несмотря на то что при выполнении программы часто обнаруживаются синтаксические ошибки, мы в данной классификации ограничимся такими условиями, как "переполнение" или "неверное преобразование типов данных".

Синтаксические и логические ошибки можно устранить на стадии разработки программы. Сложнее обстоит дело с ошибками последней категории. При попытке пользователя присвоить целой переменной значение 10^999 программа прекратит работу и выдаст сообщение об ошибке "Overflow" (переполнение). То же самое произойдет при вычислениях, результат которых выходит за пределы разрешенного диапазона. Однако было бы еще хуже, если бы программа, содержащая подобные ошиб-

ки, не останавливалась, а ее текст самопроизвольно изменялся бы, в особенности если человек, регулярно использующий программу, не является ее автором. В MSX-Бейсике, как правило, ошибки обнаруживаются и обрабатываются с помощью особых программ, а не интерпретатора.

Каждому возможному типу ошибки соответствует номер кода, например под номером 2 в списке значится уже знакомая нам "Syntax error" (синтаксическая ошибка). При появлении ошибки ее код записывается в специальную переменную ERR, а номер строки, содержащей недопустимый оператор, – в переменную ERL.

Прерывания. Итак, существуют средства обнаружения фрагментов программы, выполняющихся не так, как запланировано, или совсем не выполняющихся. Однако программисту на самом деле важнее знать, когда (а не где) возникла ошибка, чтобы принять соответствующие меры. Можно было бы регулярно опрашивать состояние некоторых переменных, но это разумно лишь для очень коротких программ. Вместо этого используется особый тип команд ветвления – прерывания.

Если какой-то ситуации в программе должно соответствовать прерывание, MSX-Бейсик автоматически анализирует появление такой ситуации в начале каждой выполняемой программной строки, что значительно удобнее и быстрее, чем опрос.

Когда "ловушка" срабатывает, т.е. проверяемое условие фиксируется, происходит автоматический переход к участку программы, обрабатывающему это условие. Данный раздел программы обычно так и называется – подпрограмма обработки прерывания. При таком способе поиск ошибки инициируется оператором ON ERROR GOTO. Оператор содержит номер строки, с которой начинается подпрограмма обработки прерывания, и включается в программу лишь один раз, обычно вначале.

Подпрограмма обработки прерывания должна заканчиваться одним из двух операторов – END либо RESUME. Второй из них делает возможным продолжение вычислений после обработки ошибки и имеет много общего с оператором RETURN. Он может иметь одну из следующих форм:

RESUME или RESUME 0	Вычисления продолжаются со строки, где встретилась ошибка:
RESUME NEXT	Вычисления продолжаются со строки, непосредственно следующей за строкой, содержащей ошибку;
RESUME <номер строки>	Вычисления продолжаются со строки с указанным номером.

Программа 4.8 иллюстрирует организацию ловушки для ошибки. В данном случае отслеживается условие "переполнение", т.е. превышение числом установленной верхней границы.

Программа 4.8

```
10 REM •
20 REM • Обработка ошибки
30 REM •
40 ON ERROR GOTO 140
```

```
50 CLS
60 WIDTH 38
70 INPUT "Введите целое число":A%
80 PRINT "Число":A%:"внутри диапазона,"
90 PRINT "допустимого для целых чисел"
100 END
110 REM *
120 REM * Обработка ошибки
130 REM *
140 IF ERR=6 THEN PRINT "!!ПЕРЕПОЛНЕНИЕ!!":
PRINT "Число вне допустимых границ"
150 RESUME
```

Организация прерываний по нестандартным ошибкам. При установке ловушек можно не ограничиваться стандартными сообщениями MSX-Бейсика об ошибках. Если в вашей программе требуется, чтобы вводимое число находилось в интервале между 0 и 1000, нарушение этого требования можно рассматривать как ошибку, которой присваивается собственный код. В настоящее время в MSX-Бейсике задействованы не все номера кодов. Так, коды 26–49 и 60–255 находятся в распоряжении программиста. (В последующих версиях MSX-Бейсика эти значения могут изменяться.)

Для подпрограммы обработки указанной ошибки ввода возьмем максимальный код. Соответствующие операторы могут иметь такой вид:

```
10 ON ERROR GOTO 1000
...
50 IF X < 0 OR X > 1000 THEN ERROR 255
...
1000 IF ERR=255 THEN PRINT "Число вне диапазона"
1010 RESUME
```

Строка 50 при обнаружении ошибки вызывает автоматический переход к подпрограмме.

Формирование собственных сообщений об ошибках приносит не так уж много пользы. Этот способ несколько неуклюж – он требует наличия двух операторов IF...THEN, тогда как фактически достаточно одного. Главное его преимущество состоит в том, что появляется возможность сгруппировать в программе все сообщения об ошибках и сделать их единообразными, что облегчает процедуру пополнения списка обрабатываемых ошибок.

ПРЕРЫВАНИЯ ПО ТАЙМЕРУ

Периодические колебания, отсчитываемые переменной TIME, могут послужить источником прерываний еще одного типа. Если включить режим этого прерывания, то обращение к соответствующей подпрограмме обработки произойдет через строго определенный интервал времени. Включение выполняется оператором INTERVAL ON. Адрес перехода определяется в соответствии с командой ON INTERVAL. В качестве

единицы отсчета времени выбирается промежуток в 1/50 секунды ("тик"). Таким образом,

ON INTERVAL = 100

будет вызывать обращение к подпрограмме обработки через каждые две секунды (при условии, что предварительно был выполнен оператор INTERVAL ON). Прерывания этого типа можно отключить или приостановить с помощью следующих двух операторов: INTERVAL OFF, который полностью отключает режим прерывания, и INTERVAL STOP, который продолжает периодически анализировать условие, но результат обработки непосредственно не проявляется. Лишь встретив следующий оператор INTERVAL ON, MSX-Бейсик "вспоминает", появилось ли условие прерывания, после чего немедленно выполняется подпрограмма обработки.

Обычно оператор INTERVAL STOP выполняется самим интерпретатором на входе в подпрограмму обработки прерываний и автоматически отменяется оператором INTERVAL ON при выходе из нее.

В программе 4.9 счетчик времени используется для подачи звукового сигнала (оператор BEEP) через каждые 10 секунд.

Программа 4.9

```
10 REM *
20 REM * Прерывание по таймеру
30 REM *
40 ON INTERVAL=500 GOSUB 110
50 INTERVAL ON
60 CLS : PRINT "10-сек. таймер"
70 GOTO 70
80 REM *
90 REM * Подпрограмма установки интервала
100 REM *
110 BEEP : PRINT "Интервал 10 секунд"
120 RETURN
```

Подобная структура (ON/STOP/OFF) является общей и для всех последующих операторов обработки прерываний.

ПРЕРЫВАНИЯ ПО СИГНАЛАМ ДЖОЙСТИКА

Обработка прерываний от кнопок джойстика обеспечивается оператором STRIG(<N>) ON, где N – номер пусковой кнопки. Выбор подпрограммы обработки прерывания регулируется оператором

ON STRIG GOSUB <номер строки>[,<номер строки>]...

Этот оператор действует подобно оператору ON GOSUB. Переход к первому номеру строки из списка осуществляется в том случае, если нажата клавиша пробела, ко второму – если нажата кнопка 1 джойстика 1 и т.д. Всего допускается до пяти адресов перехода.

Здесь также имеются соответствующие команды STRIG(<N>) STOP и STRIG(<N>) OFF. В программе 4.10 прерывания по STRIG и INTERVAL используются для моделирования работы простого реле времени.

Программа 4.10

```
10 REM *
20 REM * Прерывание по INTERVAL
30 REM * Реле времени
40 REM *
50 ON STRIG GOSUB 270
60 CLS
70 R=RND(-TIME)
80 PRINT "Реле времени"
90 PRINT
100 PRINT "Когда раздастся звонок,"
110 PRINT "нажмите пробел"
120 PRINT
130 T=INT(RND(1)*500)+10
140 ON INTERVAL = T GOSUB 200
150 INTERVAL ON
160 GOTO 160
170 REM *
180 REM * Подпрограмма отсчета времени
190 REM *
200 INTERVAL OFF
210 BEEP:TIME=0
220 STRIG(0) ON
230 GOTO 230
240 REM *
250 REM * Анализ прерывания от джойстика
260 REM *
270 STRIG(0) OFF
280 PRINT "Reaction Time"
290 PRINT "=";TIME/50:"seconds"
300 END
```

ПРЕРЫВАНИЯ ПО СИГНАЛАМ КЛАВИАТУРЫ

При обработке прерываний от функциональных клавиш ситуация во многом сходна с рассмотренными выше. Оператор KEY(<N>) активизирует режим ловушки для определенной клавиши, а ON KEY GOSUB обеспечивает переход к подпрограмме обработки. Отметим, что команды KEY ON и KEY OFF не имеют отношения к обслуживанию прерываний: их назначение заключается всего лишь в том, что они включают и выключают на экране строку подсказки для функциональных клавиш. В операторе ONKEY GOSUB можно указывать до 10 адресов перехода.

ПРЕРЫВАНИЯ ПО КОМБИНАЦИИ КЛАВИШ CTRL-STOP

При одновременном нажатии клавиш CTRL и STOP нормальное выполнение программы прекращается. Этого можно избежать, пользуясь прерыванием по ON STOP. Его основное назначение — защита программы от случайного вмешательства. В любую разрабатываемую программу этот оператор нужно включать в последнюю очередь. Если им пользоваться невнимательно, можно "добиться" того, что программу будет невозможно остановить без выключения компьютера, например:

```
10 ON STOP GOSUB 1000
...
1000 RETURN
```

ЗАМЕЧАНИЯ

В том случае, когда одновременно включены режимы прерывания для функциональной клавиши 1 и кнопки 0 джойстика, при работе функции STICK(0) может возникнуть неожиданное отклонение от нормы. В ряде случаев MSX-Бейсик может интерпретировать одновременное нажатие пробела и клавиши управления курсором как нажатие функциональной клавиши 1.

Оператор ON ERROR GOTO всегда сбрасывает все установленные к этому времени прерывания. Лучше всего в подпрограмме обработки ошибок предусмотреть специальные средства для отключения прерываний.

Поскольку действие прерываний внешне незаметно, ими нужно пользоваться очень аккуратно, особенно в больших программах с разнообразными типами и условиями прерываний.

СВОДКА СЛУЖЕБНЫХ СЛОВ

FOR <счетчик цикла> = <начальное значение> TO <конечное значение> [STEP <приращение>]

Функции

INKEY$, STICK(<N>), STRIG(<N>)

Кроме того, см. в приложении I:

PAD(<N>), PDL(<N>)

Прерывания

По ошибкам

ON ERROR GOTO <номер строки>

ERROR = <код ошибки>

RESUME [0]

RESUME NEXT

RESUME <номер строки>

END

ERR, ERL

По таймеру
ON INTERVAL = <промежуток времени • 1/50 с>
GOSUB <номер строки>
INTERVAL ON | STOP | OFF

По сигналам джойстика
ON STRIG GOSUB <номер строки>[,< >][...]
STRIG(<N>) ON | STOP | OFF

По сигналам клавиатуры
ON KEY GOSUB <номер строки>[,<номер строки>][...]
KEY (<N>) ON | STOP | OFF

По комбинации клавиш CTRL-STOP
ON STOP GOSUB <номер строки>
STOP ON | STOP | OFF

Глава 5. ВВОД, ВЫВОД И ОБРАБОТКА СТРОК

Операторы INPUT и PRINT относятся к числу наиболее часто употребляемых операторов Бейсика. Несмотря на удобство в использовании, их нельзя назвать достаточно изящными и элегантными. В настоящей главе описываются разнообразные способы получения данных извне и гибкого изменения формы выводимой информации.

В MSX-Бейсике имеется много функций, связанных с обработкой строковых данных. Эти функции и работа с ними будут подробно описаны ниже.

НАБОР СИМВОЛОВ MSX-БЕЙСИКА

Каждому символу MSX-Бейсика присвоено однозначно определенное число, которое называется его *кодом ASCII*, например буква A (латинская) имеет код 65, а знак "+" – код 43. Код ASCII позволяет обмениваться информацией между различными компьютерными системами: так, 65 всегда обозначает букву A в любом компьютере, использующем этот код. Код ASCII содержит 128 символов. Номера с 0 по 31 имеют специальное назначение и называются управляющими кодами, например код 13 соответствует клавише RETURN.

В MSX-Бейсике применяется более 128 символов. Таким образом, среди них есть коды, не относящиеся к стандарту ASCII. Они имеют номера с 129 по 255 и включают символы, которыми располагает не каждый ASCII-компьютер, в частности специальные графические символы.

Если вам нужно узнать номер кода, соответствующего определенному символу, можно воспользоваться функцией ASC( ). Она возвращает значение кода ASCII для первого символа в строке. Например,

```
                          Результат
PRINT ASC("ZAPHOD")       90 (код "Z")
PRINT ASC("*")            42
```

Если в качестве аргумента функции ASC( ) взять пустую строку, на экране появится сообщение об ошибке.

Функцией, дополняющей ASC( ), является CHR$( ). По данному коду CHR$( ) возвращает соответствующий символ. В результате выполнения команды

```
PRINT CHR$(7)
```

(управляющий код) раздается звонок, а по команде CHR$(33) на экране изображается восклицательный знак. Некоторым символам MSX-Бейсика соответствуют двойные коды. Для их получения следует к CHR$(1) добавить значение CHR$( ) с аргументом от 65 до 95, например:

	Результат
CHR$(1) + CHR$(65)	рожица
CHR$(1) + CHR$(78)	символ музыкальной ноты

Если символ, имеющий двойной код, задать в качестве аргумента функции ASC( ), вы всегда получите единицу.

Если вы не располагаете специальным принтером MSX, лучше включать нестандартные символы ASCII в программы, работающие с функцией CHR$( ), а не пользоваться строковыми константами. Программа 5.1 выдает все символы MSX-Бейсика, полученные с помощью функции CHR$( ). Результаты ее работы представлены на рис. 5.1.

Программа 5.1

```
10 REM •
20 REM • Набор символов
30 REM •
40 KEY OFF
50 COLOR 15,4,4
60 SCREEN 1
70 PRINT "Двойные коды"
80 PRINT"
90 FOR I=65 TO 95
100 PRINT CHR$(1)+CHR$(I);
110 NEXT
120 PRINT : PRINT
130 PRINT "Одинарные коды"
140 PRINT
150 FOR I = 32 TO 255
160 PRINT CHR$(I);
170 NEXT
```

ОПЕРАЦИИ НАД СТРОКАМИ

Ранее была описана лишь одна из операций над строками — конкатенация, или слияние двух строк. С помощью строковых функций MSX-Бейсика можно программным путем урезать строки, выделять их

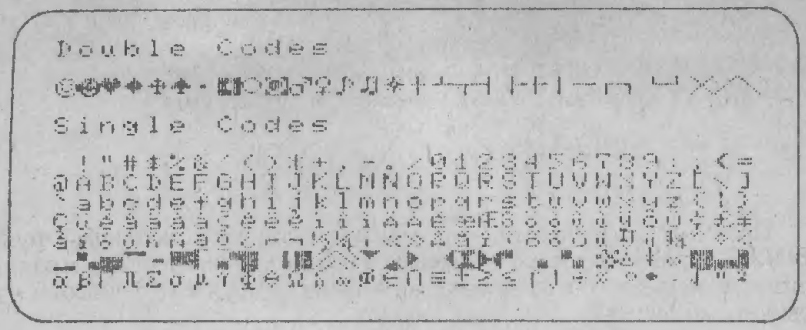

Рис. 5.1. Набор символов MSX-Бейсика (результат выполнения программы 5.1)

фрагменты, менять местами и т.д. Первым параметром строки, который мы можем определить, является ее длина. Соответствующая функция имеет название LEN( ). Она подсчитывает число символов в строке, включая невидимые управляющие символы, обеспечивающие, например, звонок или возврат на одну позицию. Если LEN( ) применяется к пустой строке, результатом будет нуль (программа 5.2).

Программа 5.2

```
10 REM *
20 REM * Функция LEN( )
30 REM *
40 KEY OFF
50 CLS
70 PRINT "Введите строку:"
80 INPUT T$
90 PRINT
100 PRINT "Строка содержит"; LEN(T$)
110 PRINT " знаков."
```

Функции LEFT$( ) **и** RIGHT$( ). Эти функции возвращают заданное число символов, начиная соответственно от левого или от правого конца строки. Для них действуют одинаковые правила синтаксиса:

LEFT$(<строка>,<число возвращаемых символов с левого края>)
RIGHT$(<строка>,<число возвращаемых символов с правого края>)

Если заданное число возвращаемых символов превышает длину строки, последняя выдается целиком. Приведем четыре примера выполнения этих функций:

	Результат
LEFT$("MSX Computers",3)	"MSX"
RIGHT$("MSX Computers",9)	"Computers"
RIGHT$("Hiccup",200)	"Hiccup"
LEFT$("Yes",1)	"Y"

Программа 5.3 запрашивает число, создает с помощью функции BIN$( ) строку двоичных символов, а затем меняет местами младшие и старшие четверки битов полученного числа. Результат ее работы можно увидеть на рис. 5.2.

Программа 5.3

```
10 REM *
20 REM * Обмен полубайтами
30 REM *
40 KEY OFF
50 SCREEN 0
60 INPUT "Введите число от 0 до 255":N
70 IF N<0 OR N>255 THEN GOTO 60
80 BYTE$ = BIN$(N)
90 REM *
100 REM * Если битов меньше восьми,
110 REM * добавляются ведущие нули
120 REM *
130 IF LEN(BYTE$)<8 THEN BYTE$="0"+BYTE$
:GOTO 130
140 REM *
150 REM * Перестановка полубайтов
160 REM *
170 H$ = LEFT$(BYTE$,4)
180 L$ = RIGHT$(BYTE$,4)
190 NBYTE$ = L$ + H$
200 PRINT
210 PRINT "Исходное значение"
220 PRINT
230 PRINT "(двоичное): ":BYTE$
240 PRINT
250 PRINT "Новое значение"
260 PRINT
270 PRINT "(двоичное): ":NBYTE$
280 END
```

```
Введите число от 0 до 255 ? 97

Исходное значение :

(двоичное) :  01100001

Новое значение :

(двоичное) :  00010110
```

Рис. 5.2. Обмен полубайтами (программа 5.3)

Поиск и замена символов. Функция INSTR( ) позволяет проанализировать строку символов на предмет вхождения в нее другой строки. Например, можно выяснить, содержится ли в данной строке фрагмент "шар". При обнаружении соответствия (т.е. когда строка включает такой набор символов) функция INSTR( ) возвращает номер начальной позиции проверяемого фрагмента. Если одна строка не содержит другую, INSTR( ) возвращает нуль:

```
                                                    Результат
PRINT INSTR("Земля имеет форму шара","шар")            19
PRINT INSTR("Все эльфы зеленые","дикобразы")            0
```

Можно также указать номер позиции, с которой должен начинаться просмотр и анализ строки. Эта величина должна находиться между 1 и 255.

```
                                                    Результат
PRINT INSTR(7, "Привет, мартышка", "Привет")            0
PRINT INSTR(7, "Привет, мартышка", "ты")               12
```

Перемещая точку начала просмотра, можно выявить все вхождения одного фрагмента в другой и сосчитать их. Программа 5.4 и рис. 5.3 демонстрируют процесс подсчета слов и результат.

Программа 5.4

```
10 REM *
20 REM * Подсчет слов
30 REM *
40 KEY OFF
50 SCREEN 0
60 WIDTH 38
70 PRINT "Введите строку:"
80 PRINT
```

```
90 INPUT T$
100 IF LEN(T$)=0 THEN GOTO 70
110 PRINT "Какое слово искать:"
120 INPUT CWD$
130 REM *
140 REM * Анализ и подсчет слов
150 REM *
160 I = 1 : C=0
170 X = INSTR(I,T$,CWD$)
180 IF X=0 THEN 230
190 C = C+1
200 I = X+1
210 GOTO 170
220 PRINT
230 PRINT
240 PRINT "Слово: ";CWD$
250 PRINT "Число повторений:";C
260 END
```

```
Введите строку:

? Шалтай Болтай сидел на стене,
Шалтай Болтай свалился во сне.
Какое слово искать :
? Шалтай

Слово : Шалтай
Число повторений : 2
```

Рис.5.3. Подсчет слов в программе 5.4

Фрагмент строки можно выделить с помощью функции MID$( ). Синтаксис ее таков:

MID$(A$,X[,Y])

Здесь A$ обозначает обрабатываемую строку, X – начальную позицию внутри нее, а Y – длину выделяемого фрагмента. Если последний аргумент опущен, выдаются все остающиеся символы от начальной точки до правого конца строки. Действие этой функции можно проследить на следующих примерах:

	Результат
PRINT MID$("Большие змеи вызывают страх",9)	"змеи вызывают страх"
PRINT MID$("Фред работает хирургом",6,8)	"работает"

В программе 5.5 с помощью функции MID$( ) анализируется строка символов, после чего все строчные буквы заменяются соответствующими прописными. Результат показан на рис. 5.4.

Программа 5.5

```
10 REM •
20 REM • Переход к заглавным буквам
30 REM •
40 KEY OFF
50 SCREEN 0
60 WIDTH 38
70 PRINT "Введите текст:"
80 PRINT
90 INPUT T$
100 L = LEN(T$)
110 IF L=0 THEN GOTO 90
120 N$ = ""
130 FOR P = 1 TO L
140 S$ = MID$(T$,P,1)
150 IF S$<"a" OR S$>="z" GOTO 240
160 REM •
170 REM • Замена на заглавную
180 REM •
190 S = ASC(S$)-32
200 S$ = CHR$(S)
210 REM •
220 REM • Формирование новой строки
230 REM •
240 N$ = N$ + S$
250 NEXT P
260 PRINT
270 PRINT "Старый текст: " : PRINT
280 PRINT T$
290 PRINT
300 PRINT "Новый текст: " : PRINT
310 PRINT N$
320 END
```

Можно также использовать MID$ в качестве оператора. В таком варианте один фрагмент строки заменяется другим. Оператор MID$ имеет следующий формат:

MID$(<строковая переменная>,<начальная позиция>[,<длина подстроки>] = <заменяющая подстрока>

Если вы хотите в предложении "Все деревья зелены" заменить слово "деревья" словом "лягушки", можно обратиться к программе 5.6.

```
Введите текст:

? Some of this TEXT is lower case.

Старый текст:
Some of this TEXT is lower case.

Новый текст:
SOME OF THIS TEXT IS LOWER CASE.
```

Рис. 5.4. Замена строчных букв прописными в программе 5.5

Программа 5.6

```
10 REM *
20 REM * Замена фрагмента строки
30 REM *
40 A$ = "Все деревья зелены"
50 B$ = "лягушки"
60 PRINT A$
70 MID$(A$,5,7)=B$
80 PRINT A$
```

Отметим, что длина исходной строковой переменной в любом случае остается прежней. Если попытаться заменить слово "деревья" строкой "огромные ушастые слоны", то в результате получится:

"Все огромные ушаст".

В этом состоит определенное неудобство использования MID$ в качестве оператора для замещения фрагментов строк: замене подлежат лишь строки равной длины. С помощью других строковых функций такое ограничение легко обойти. Программа 5.7 позволяет выполнять удаление, вставку и замену фрагментов текста. Ее работа иллюстрируется рис. 5.5.

Программа 5.7

```
10 REM *
20 REM * Вставка, удаление, замена
30 REM *
40 KEY OFF
50 SCREEN 0
60 WIDTH 38
70 CLEAR 400
80 PRINT "Редактирование строки"
90 PRINT "----------------"
```

```
100 PRINT
110 PRINT "Введите текст:"
120 INPUT TEXT$
130 L=LEN(TEXT$)
140 IF L=0 THEN GOTO 120
150 REM •
160 REM • Главный цикл
170 REM •
180 PRINT
190 PRINT "----------"
200 PRINT "1 – вставка":PRINT "2 – удаление
":PRINT "3 – замена"
210 PRINT "----------"
220 PRINT
230 PRINT "Текст: ":TEXT$ : PRINT
240 INPUT "Номер функции":O
250 IF O<1 OR O>3 THEN GOTO 200
260 PRINT
270 ON O GOSUB 320,470,600
280 GOTO 190
290 REM •
300 REM • Подпрограмма вставки
310 REM •
320 INPUT "Вставить текст:";I$
330 IL=LEN(I$) : IF IL=0 THEN 320
340 IF IL+L>255 THEN PRINT "• Нет места"
: RETURN
350 PRINT
360 INPUT "Вставить с позиции ";P
370 IF P<1 OR P>LEN(TEXT$) THEN 360
380 PRINT
390 IF P>L+1 THEN GOTO 360
400 A$ = LEFT$(TEXT$,P–1)
410 B$ = RIGHT$(TEXT$,L–P+1)
420 TEXT$= A$ + I$ + B$
430 L = LEN(TEXT$)
440 RETURN
450 REM •
460 REM • Подпрограмма удаления
470 REM •
480 INPUT "Удалить текст:";D$
490 PRINT
500 DL=LEN(D$) : IF DL=0 THEN 480
510 P = INSTR(TEXT$,D$)
520 IF P=0 THEN PRINT "• Не найден"
:PRINT : RETURN
530 A$=LEFT$(TEXT$,F–1)
540 B$=RIGHT$(TEXT$,L–P–DL+1)
```

```
Редактирование строки
---------------------

Введите текст:
? Вот кусок текста

---------------------
1 – вставка
2 – удаление
3 – замена
---------------------

Текст: Вот кусок текста

Функция? 3

Найти текст: ? кусок
Заменить на текст: ? образец
---------------------

1 – вставка
2 – удаление
3 – замена
---------------------

Текст:Вот образец текста

Функция? 2

Удалить текст: ? пингвин

• Не найден
---------------------
1 – вставка
2 – удаление
3 – замена
---------------------

Текст : Вот образец текста
```

Рис. 5.5. Результат выполнения программы 5.7

```
550 TEXT$=A$+B$
560 L=LEN(TEXT$)
570 RETURN
580 REM •
590 REM • Подпрограмма замены
```

```
600 REM *
610 INPUT "Найти текст.";S$
620 SL=LEN(S$) : IF SL=0 THEN 610
630 P = INSTR(TEXT$,S$)
640 IF P=0 THEN PRINT "* Не найден" : PR
INT : RETURN
650 PRINT
660 INPUT "Заменить на текст.";R$
670 RL=LEN(R$) : IF RL=0 THEN 670
680 PRINT
690 REM *
700 REM * Замена строк равной длины
710 REM *
720 IF RL=SL THEN MID$(TEXT$,P,RL)=R$ :
RETURN
730 IF RL>SL GOTO 870
740 REM *
750 REM * Замена длинной строки
760 REM * на короткую
770 REM *
780 MID$(TEXT$,P,RL)=R$
790 P=P+RL : DL=SL-RL
800 GOSUB 530 : RETURN
810 REM *
820 REM * Замена короткой строки
830 REM * на длинную
840 REM *
850 B = RL-SL
860 IF B+L>255 THEN PRINT "* Нет места"
:RETURN
870 MID$(TEXT$,P,SL)=R$
880 P = P+SL
890 A$ = LEFT$(TEXT$,P-1)
900 B$=RIGHT$(TEXT$,L-P+1)
910 TEXT$=A$+RIGHT$(R$,B)+B$
920 L = LEN(TEXT$)
930 RETURN
```

ВАРИАНТЫ ОПЕРАТОРА INPUT

Одно из неудобств при использовании оператора INPUT состоит в том, что в качестве приглашения к вводу информации неизменно возникает вопросительный знак. Вместо INPUT можно применять другой оператор – LINE INPUT, который не вызывает автоматического появления вопросительного знака. Синтаксический формат LINE INPUT напоминает формат INPUT, но допускает ввод с клавиатуры до 254 символов. Вводимые данные сразу отображаются на экране. Для окончания ввода нужно нажать клавишу RETURN. Основное ограничение здесь

заключается в том, что с помощью этого оператора вводятся только строковые данные. При вводе числовой информации одного оператора LINE INPUT недостаточно.

Встроенная функция MSX-Бейсика VAL( ) преобразует формат числа из строкового в обычный числовой. Так, результатом выполнения функции VAL("121.62") является число 121.62. При вводе пустой строки как аргумента значением функции VAL( ) всегда будет нуль. В процессе преобразования все начальные пробелы и управляющие символы игнорируются.

Главное назначение функции VAL( ) состоит в переводе строковых выражений, соответствующих двоичным, шестнадцатиричным или восьмиричным величинам, в десятичную систему. Например, VAL("&B10000001") возвращает в результате число 129.

Дополнительно к VAL( ) применяется функция STR$( ), преобразующая числовой формат в строковый. При печати обычного числа слева от первой цифры остается пробел. Если же перед печатью число преобразовать в строку, этот пробел можно исключить с помощью функций RIGHT$( ) или MID$( ). Такой прием позволяет сделать при выводе текстовую и числовую информацию более однородной. В программе 5.8 показан пример совместного выполнения VAL( ), STR$( ) и LINE INPUT.

Программа 5.8

```
10 REM *
20 REM * VAL и LINE INPUT
30 REM *
40 LINE INPUT "Значение X = ";A$
50 LINE INPUT "Значение Y = ";B$
60 PRINT
70 X = VAL(A$)
80 Y = VAL(B$)
90 P = X*Y
100 PRINT A$;CHR$(42);B$;" =";P
110 PRINT
120 P$=STR$(P),
130 PRINT "Старший разряд произведения: ";
140 PRINT MID$(P$,2,1)
150 END
```

Еще один способ ввода данных с клавиатуры связан с применением функции INPUT$( ), которая принимает с клавиатуры заданное число символов и присваивает их строковой переменной. В этом случае вводимые символы не дублируются на экране, и можно обойтись без обязательного в других ситуациях указателя окончания ввода — RETURN. Оператор A$ = INPUT$(4) присваивает первые четыре введенных символа переменной A$. Несложная программа 5.9, анализирующая некоторый пароль, демонстрирует один из вариантов применения этого оператора, хотя в качестве реальной системы защиты она, разумеется, оставляет желать лучшего.

Программа 5.9

```
10 REM *
20 REM * Пароль
30 REM *
40 CODE$ = "SECRET": COUNT=0
50 PRINT "Имя: "
60 INPUT N$
70 PRINT "Пароль: "
80 REM *
90 REM * Чтение шести символов
100 REM *
110 P$=INPUT$(6)
120 IF P$=CODE$ THEN GOTO 150
130 COUNT=COUNT+1
140 IF COUNT < 3 THEN GOTO 70 ELSE PRINT
"Проход закрыт": END
150 PRINT
160 PRINT N$
170 PRINT "Проходи, приятель"
180 END
```

Значительно полезнее функция INPUT$( ) в ситуации, когда нужно принимать данные с клавиатуры по одному символу. Если в ходе выполнения программы требуется ввести число, с помощью INPUT$( ) можно проверить его по цифрам. В принципе можно считать, что любое вводимое число составлено из следующих символов:

* знака "+" или "–" в том случае, если это первый символ;
* десятичной точки (встречается лишь один раз);
* цифр от 0 до 9.

Предполагается, что ведущие пробелы игнорируются (это всегда можно обеспечить использованием VAL( )) и в середине числа пробелы не встречаются. Число воспринимается полностью введенным при нажатии клавиши RETURN. Примером программы, осуществляющей такой посимвольный контроль ввода, может служить программа 5.10. Сферу применения этой программы можно расширить, предусмотрев в ней возможность анализа чисел в формате с плавающей точкой, а также проверки нахождения числа в нужном диапазоне. Кроме того, можно задействовать клавиши INS и DEL, что позволяет вносить исправления во вводимые данные.

Программа 5.10

```
10 REM *
20 REM * Контроль ввода чисел
30 REM *
40 CLS
50 N$ = "": P=0
```

```
60 PRINT "Введите число: ";
70 A$ = INPUT$(1)
80 IF A$= CHR$(13) THEN 180
90 IF A$= "+" AND N$="" THEN 150
100 IF A$="-" AND N$="" THEN 150
110 IF A$= "." THEN P=P+1
120 IF P= 1 THEN 150
130 IF A$>="0" AND A$<="9" THEN 150
140 BEEP : GOTO 70
150 N$=N$+A$
160 PRINT A$;
170 GOTO 70
180 PRINT : PRINT
190 PRINT "Введенное число ";VAL(N$)
200 END
```

Функция INPUT$( ) несколько отличается от INKEY$. Значение INKEY$ необходимо постоянно проверять до тех пор, пока не встретится нуль, в то время как INPUT$ автоматически обращается к клавиатуре столько раз, сколько требуется для ввода нужного числа символов.

ПОЗИЦИОНИРОВАНИЕ КУРСОРА

Хорошо знакомый вам атрибут экрана, курсор, можно перемещать в любую точку с помощью команды LOCATE. Текстовый экран насчитывает 40 * 24 позиции в варианте SCREEN 0 и 32 * 24 позиции для SCREEN 1. Задавая команду LOCATE со значениями пары координат, можно изменять местоположение курсора и выводить данные в нужное место на экране.

Координаты, необходимые для выполнения команды LOCATE, определяют положение точки на экране по вертикали и по горизонтали. При вводе команды необходимо иметь в виду основное соглашение о том, что точке 0,0 соответствует верхний левый угол экрана. Если перед этим выполнялась команда WIDTH, предельные значения координат, допустимые для LOCATE, могут измениться. Так, после выполнения команды WIDTH 20 попытка поместить курсор в позицию 25,20 вызовет ошибку. В отсутствие WIDTH ширина экрана по умолчанию принимается равной стандартному значению для данного экрана. Программа 5.11 выводит текст в различные точки на экране.

Программа 5.11

```
10 REM *
20 REM * Оператор LOCATE
30 REM *
40 X = RND(-TIME)
50 SCREEN 0
60 KEY OFF : WIDTH 40
70 FOR I = 1 TO 20
```

```
80 X=INT(RND(1)*35)
90 Y=INT(RND(1)*23)
100 LOCATE X,Y,1
110 PRINT "Привет";
120 NEXT
```

По ключу в конце списка параметров команды LOCATE курсор может быть выключен:

0	выключение курсора
1	включение курсора

Имеются две функции, позволяющие определять положение курсора на экране: CSRLIN, возвращающая координату курсора по вертикали, и POS(0) – по горизонтали. Аргумент функции POS( ) может быть любым, поскольку он при работе игнорируется. Программа 5.12 позволяет наряду с вводом текста организовать отсчет времени (часы) в правом верхнем углу экрана. Перед изменением текущего значения времени считываются значения CSRLIN и POS( ), что позволяет запомнить координаты текущего положения курсора и восстановить его. Заметим, что "опрос" клавиатуры выполняется с помощью INKEY$. В этом случае функция INPUT$( ) неприменима, так как она ожидает нажатия клавиши, поэтому ввод с клавиатуры будет мешать нормальному изменению состояния счетчика времени.

Программа 5.12

```
10 REM *
20 REM * Часы
30 REM *
40 CLS : WIDTH 40
50 ON INTERVAL=50 GOSUB 230
60 TIME = 0
70 INTERVAL ON
80 LOCATE 16,0,0 : PRINT "Время:"
90 LOCATE 24,0 : PRINT H;":";M;":";S
100 INTERVAL ON
110 A$=INKEY$ : IF A$="" THEN GOTO 110
120 PRINT A$;
130 REM *
140 REM * Очистка заполненного экрана
150 REM *
160 IF CSRLIN=23 AND POS(0)=37 THEN
PRINT A$:CLS:LOCATE 0,1,0:GOTO 80
170 GOTO 110
180 REM *
190 REM * Подпрограмма прерывания
200 REM *
210 REM * Запоминание координат курсора
```

```
220 REM *
230 X=POS(0) : Y = CSRLIN
240 S=S+1
250 IF S = 60 THEN M = M+1 : S=0
260 IF M=60 THEN H=H+1 : M=0
270 IF H=24 THEN H=0 : LOCATE 24,0,0
:PRINT"
280 LOCATE 24,0,0 : PRINT H;":";M;":";S
290 LOCATE X,Y,1
300 RETURN
```

ПРОЧИЕ СТРОКОВЫЕ ФУНКЦИИ

Список вывода команды PRINT можно дополнять некоторыми функциями. Первой из них рассмотрим TAB( ). Эта функция перемещает курсор на заданное число позиций по горизонтали от левого края экрана. Величина сдвига задается целым аргументом в пределах от 0 до 255. Если значение аргумента превышает длину строки, курсор спускается на следующую строку.

Другая функция, выполняющаяся вместе с командой PRINT, – SPC( ). Она выводит заданное число пробелов. Отметим, что TAB( ) лишь изменяет положение курсора, в то время как SPC( ) генерирует новые символы, и соответственно перемещает курсор. Это иллюстрируется программой 5.13 и рис. 5.6.

Программа 5.13

```
10 REM *
20 REM * Функции SPC и TAB
30 REM *
40 SCREEN 0
50 KEY OFF : WIDTH 38
60 FOR I = 0 TO 5
70 PRINT TAB(I);"*";
80 NEXT
90 PRINT
100 FOR I = 0 TO 5
110 PRINT SPC(I);"*";
120 NEXT
```

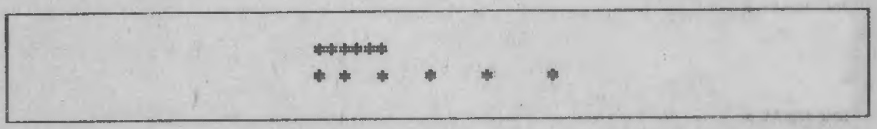

Рис. 5.6. Действие функций SPC и TAB (программа 5.13)

Функция STRING$ позволяет создать последовательность заданной длины, состоящую из одинаковых символов. Наряду с реальным строковым аргументом функция STRING$ может содержать его код ASCII. Так, при выполнении PRINT STRING$(35,"*") появляется строка из 35 звездочек и точно такой же результат достигается при выполнении команды PRINT STRING$(35,42). Если в качестве второго аргумента функции STRING$( ) задается строка из двух и более знаков, из нее берется лишь первый символ. Если первый элемент данной переменной определяется с помощью двойного кода, STRING$ возвращает строку, состоящую из символов, соответствующих ASCII-коду 1.

Функция SPACE$( ) создает строку заданной длины, целиком состоящую из пробелов. Все эти функции можно применять для "заполнения свободных мест" в процессе вывода данных. Программа 5.14 выводит на печать набор строк таким образом, что крайние правые символы каждой строки лежат строго друг под другом. В полиграфии это называется выключкой строки. Программа 5.15 с помощью функций STRING$( ) и TAB( ) выводит в центр экрана подчеркнутый текст. Переменная W обозначает текущую ширину экрана.

Программа 5.14

```
10 REM *
20 REM * Выравнивание строк вправо
30 REM *
40 SCREEN 0
50 KEY OFF : WIDTH 40
60 PRINT TAB(13):"Выравнивание"
70 PRINT : PRINT
80 FOR I = 0 TO 9
90 READ A$
100 FILL = 35-LEN(A$)
110 PRINT I:STRING$(FILL,"-"):A$
120 NEXT
130 DATA "The Quick Brown Fox"
140 DATA "Hellzapoppin"
150 DATA "Queen Victoria"
160 DATA "Reginald Bosanquet"
170 DATA "The Crown Jewels"
180 DATA "Mr Pye"
190 DATA "Christopher Columbus"
200 DATA "ABC"
210 DATA "Quo Vadis"
220 DATA "ELstree Studios"
```

Программа 5.15

```
10 REM *
20 REM * Центрирование и подчеркивание
30 REM *
40 W = 38 : REM * Установка ширины экрана
```

```
50 WIDTH W
60 SCREEN 0
70 LINE INPUT "Текст: ";TEXT$
80 L = LEN(TEXT$)
90 CLS
100 REM *
110 REM * Центрирование
120 REM *
130 IDT = (W-L)/2
140 PRINT TAB(IDT);TEXT$
150 REM *
160 REM * Подчеркивание
170 REM *
180 PRINT TAB(IDT);STRING$(L,195)
```

ФОРМАТИРОВАННЫЙ ВЫВОД

Одна из наиболее привлекательных возможностей MSX-Бейсика по организации вывода данных состоит в применении мощной команды PRINT USING, которой на удивление мало кто пользуется. PRINT USING позволяет получить аккуратно и единообразно оформленные результаты, содержащие как тексты, так и числа. Общий формат команды следующий:

PRINT USING <строка формата>;<список вывода>

Строка формата представляет собой выражение, которое определяет порядок вывода. Она может содержать ряд символов, одни из которых управляют выводом текстов, а другие числовой информацией. Перечислим управляющие форматоры текста:

1. ! Определяет печать первого символа строки.
2. & Позволяет вложить печатаемую строку в другую выводимую строку.
3. \<n пробелов>\ Дает возможность вывести по крайней мере два символа из строки, а также еще столько символов, сколько пробелов помещено между двумя обратными косыми чертами. Если число таких пробелов превышает число элементов в печатаемой строке, свободные места заполняются пробелами.

Программа 5.16 и рис. 5.7 демонстрируют использование этих символов формата.

Программа 5.16

```
10 REM *
20 REM * Форматоры текста
30 REM *
40 SCREEN 0
50 KEY OFF : WIDTH 38
60 PRINT
```

```
70 FOR I = 1 TO 3
80 READ A$
90 PRINT USING "!":A$
100 PRINT USING "& положение":A$
110 PRINT USING "\\":A$
120 PRINT USING "\  \":A$
130 PRINT USING "\    \":A$
140 PRINT STRING$(37,"-")
150 NEXT I
160 PRINT
170 DATA "Международное"
180 DATA "Географическое"
190 DATA "Трудное"
```

```
М
Международное положение
Ме
Межд
Междунар
-------------------------------
Г
Географическое положение
Ге
Геог
Географи
-------------------------------
Т
Трудное положение
Тр
Труд
Трудное
-------------------------------
```

Рис. 5.7. Форматированный вывод строк в программе 5.16

Приведем теперь несколько числовых форматоров и проиллюстрируем их применение на примерах.

1. # Применяется для вывода цифр. Каждая выводимая цифра соответствует одному знаку диеза; если возможен отрицательный результат, лучше предусмотрительно добавить лишний символ. Можно использовать десятичную точку. Число выводится следующим образом:

а) если слева от десятичной точки в действительности меньше цифр, чем в шаблоне формата, лишние позиции заполняются пробелами, и число сдвигается вправо. Свободные позиции слева от десятичной точки заполняются нулями;

б) если размеры числа превышают шаблон формата, выдается знак "%", указывающий на переполнение;

в) при необходимости числа справа от десятичной точки округляются в соответствии с форматом.

Более наглядно это показано в программе 5.17 и на рис. 5.8

Программа 5.17

```
10 REM *
20 REM * Числовые форматоры 1
30 REM *
40 CLS
50 PRINT TAB(1):"Исходное","Форматированное"
60 PRINT TAB(1):STRING$(8,195),STRING$
(15,195)
70 PRINT
80 FOR I = 1 TO 6
90 READ A
100 PRINT A
110 PRINT USING "###.###";A
120 NEXT
130 DATA 142.854,93.281,1.62399
140 DATA -43,-112.45,1000.83
```

Исходное	Форматированное
142.854	142.854
93.281	93.281
1.62399	1.624
-43	-43.000
-112.45	%-112.450
1000.83	%1000.830

Рис. 5.8. Программа 5.17 – форматор "# "

2. – и + Используются для указания знака печатаемого числа. "–" помещается только в конце числового поля и вызывает печать знака в этой позиции лишь для отрицательных чисел. "+" в начале или в конце числового поля задает вывод знака (как "+", так и "–"). При этом лишние позиции не печатаются, знак же помещается непосредственно перед первой цифрой (см. программу 5.18 и рис. 5.9).

Программа 5.18

```
10 REM *
20 REM * Числовые форматоры 2
30 REM *
```

```
40 CLS
50 PRINT "Форматированное"
60 PRINT STRING$(15,195)
70 PRINT
80 PRINT USING "+####.#";42.34
90 PRINT USING "+####.#";-3.357
100 PRINT USING "####.##-";434.3
110 PRINT USING "####.##-";-324.48
```

```
Форматированное
---------------

+42.3
-3.4
434.30
324.48-
```

Рис. 5.9. Программа 5.18 – форматоры "+" и "–"

3. ** Вместо ведущих пробелов обеспечивает генерацию звездочек.

4. ££ В начале числа задает печать одного символа фунта стерлингов.

5. **£ Инициирует печать звездочек вместо ведущих пробелов (при необходимости), а затем знака фунта стерлингов перед числом.

6. , Слева от десятичной точки разбивает все цифры на группы по три в каждой. Тысячи, миллионы и т.д. разделяются запятыми.

7. ^^^^ Обеспечивает запись числа в показательной("научной") форме.

Действие всех описанных выше форматоров можно проследить в программе 5.19, которая является своеобразным итогом настоящего раздела, а результат ее выполнения представлен на рис. 5.10.

Программа 5.19

```
10 REM *
20 REM * Числовые форматоры 3
30 REM *
40 CLS
50 PRINT USING "**####.##";1.453
60 PRINT
70 PRINT USING "££####,##";321.63
80 PRINT
90 PRINT USING "**£##"##";-10,3
100 PRINT
110 PRINT USING "#########.";5411284#
```

```
120 PRINT
130 PRINT USING "##.#^^^^";141223!
140 END
```

Перед указателями формата в любом из приведенных выше примеров использования формата PRINT USING могут находиться обычные текстовые данные. Вполне допустимы, в частности, такие операторы, как те, которые показаны в программе 5.20 (результат ее выполнения см. на рис. 5.11).

```
****1.45
£321.63
*-£10.30
 5,411,284
 1.4E+05
```

Рис. 5.10. Программа 5.19 – смешаные форматы

Программа 5.20

```
10 REM *
20 REM * Текст и форматоры
30 REM *
40 PRINT USING "Налог  : ###.##";92.53
50 PRINT USING "Штат   : .."; "Калифорния"
60 PRINT USING "Инициалы !!.";"Билл";" Блоггз
```

```
Налог      :  92.53
Штат       :Калиф.
Инициалы:Б.Б.
```

Рис. 5.11. Программа 5.20 – обычный текст в сочетании с форматированным

ВЫВОД НА ПРИНТЕР

Если вам настолько повезло, что ваша система снабжена принтером, то можно использовать все имеющиеся в MSX-Бейсике варианты оператора PRINT. Любые манипуляции, которые выполнялись при выводе на

экран, легко повторить и для случая с принтером. Соответствующий оператор обозначается LPRINT и допускает даже такой формат, как PRINT USING, который теперь, естественно, имеет вид LPRINT USING.

Здесь необходимо упомянуть лишь об одной функции, связанной с работой принтера,– LPOS( ). Компьютеры MSX пересылают данные гораздо быстрее, чем их в состоянии механически напечатать принтер, поэтому в ОЗУ выделяется область, называемая буфером печати. Буфер печати в процессе работы принтера пополняется данными, получаемыми из центрального процессора, которые выводятся на печать по мере освобождения устройства. LPOS(0) возвращает позицию головки принтера в буфере. (Правда, маловероятно, что вы будете часто пользоваться такой возможностью).

Принтеры представляют собой довольно сложные устройства с собственными процессором и памятью. С помощью оператора LPRINT на принтер можно подавать коды (так называемые последовательности перехода, или escape-последовательности), вызывающие, в частности, сжатие и расширение текста. Принтеры различных фирм по-разному реагируют на эти коды, поэтому за полной информацией следует обращаться к руководству по эксплуатации. Чаще всего для управления принтерами используются коды, разработанные фирмой Epson, которые в последнее время приобрели популярность среди других фирм-производителей оборудования. В настоящей книге есть несколько примеров вывода на принтер, и в каждом из них предполагается применение кодов Epson.

СВОДКА СЛУЖЕБНЫХ СЛОВ

ASC(A$), CHR$(X), CSRLIN, INPUT$(X), INSTR([X,]A$,B$),
LEFT$(A$,X), LEN(A$), LPOS(0), MID$(A$,X[,Y]), POS(0),
RIGHT$(A$,X), SPACE$(X), SPC(X), STR$(X), STRING$(X,A$),
STRING$(X,Y), TAB(X), VAL(A$)
LOCATE <X>,<Y>[,I|0>]
LPRINT [<выражение>][разделитель][,<выражение>]...
MID$ (<строковая переменная>,X[,Y])=<строковое выражение>
PRINT USING <строка формата>;<выражение>[;<выражение>]...

Глава 6. СТРУКТУРЫ ДАННЫХ

В настоящей главе описывается способ, позволяющий логически объединить набор числовых данных под общим именем. Все переменные, встречавшиеся нам прежде, можно назвать ***простыми переменными***: одному имени переменной соответствовало единственное значение. Во многих ситуациях это неудобно. Допустим, программа должна хранить и обрабатывать список из 50 названий. Для подобной программы необходимо ввести 50 независимых переменных, например N1$, N2$, N3$, ..., N50$. В таком случае, как мы знаем, переменные с именами N5$ и N50$ не будут различаться программой. При работе с таблицами число требуемых переменных резко возрастет и возникнут проблемы, связанные с их именованием.

При обработке данных, сгруппированных в списки и таблицы, гораздо удобнее ввести одно общее имя переменной. Это допускается при использовании ***массивов***.

СПИСКИ ДАННЫХ

С помощью оператора DIM список данных можно описать как массив. Оператор DIM выполняет две функции:

* определяет число элементов в списке;
* указывает тип данных для каждого элемента списка.

Чтобы описать массив, состоящий из 50 чисел с двойной точностью, можно воспользоваться следующим оператором:

```
10 DIM A!(50)
```

Как видите, переменной массива приписывается определенный тип с помощью символов описания типа. Не допускается смешивать типы данных – массив может содержать данные лишь одного типа.

К каждому элементу массива можно адресоваться по его ***индексу*** (порядковому номеру). Переменную массива A! можно рассматривать как последовательность ячеек памяти (рис. 6.1). Первый элемент массива имеет индекс "1" либо "0", поэтому обращаться к нему можно с помощью имени переменной A!(1) либо A!(0). Последний элемент описывается как A!(50) или A!(49). Наименьший допустимый индекс равен 0, а

наибольший совпадает с числом элементов в массиве. Задание индекса вне этих границ приведет к ошибке.

ПРИМЕРЫ ОБРАБОТКИ МАССИВОВ

Пусть требуется случайным образом выбрать одно из 10 имен. Это несложно сделать и без помощи массивов, но, по-видимому, такая программа будет слишком громоздкой. В программе 6.1 генерируется случайная величина, которая затем используется в качестве индекса для массива, содержащего перечень имен. В программе 6.2 также применяется датчик случайных чисел, на этот раз с целью моделирования бросания игральных костей. В массиве хранятся результаты ряда "бросаний", на основе которых вычисляется процентная частота выпадения каждой грани, т.е. очков от одного до шести (рис. 6.2).

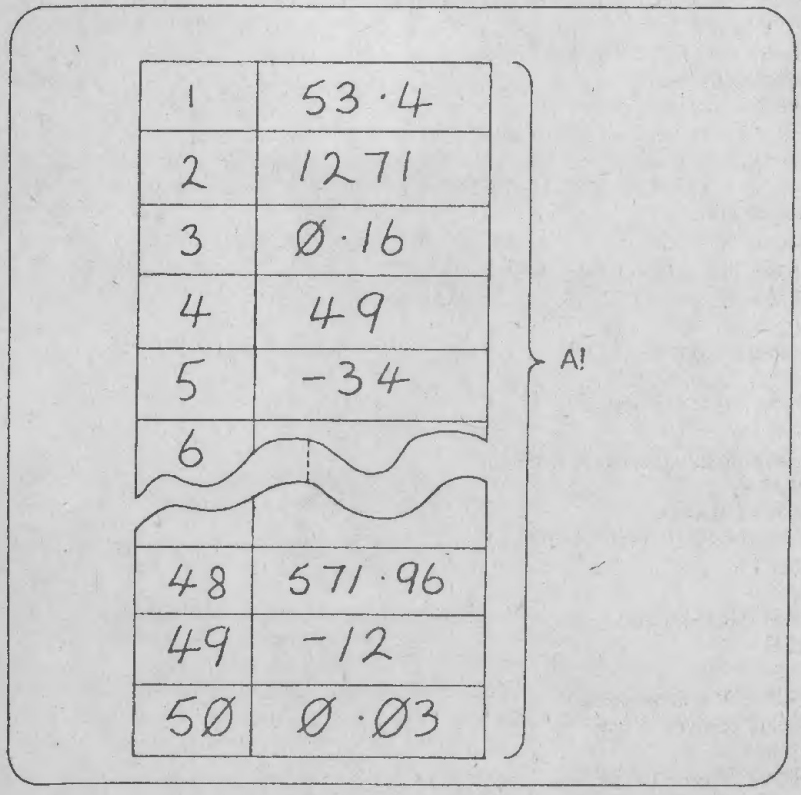

Рис. 6.1. Элементы массива A!

Программа 6.1

```
10 REM *
20 REM * Работа с массивом
30 REM *
40 CLEAR 1000
50 DIM A$(10)
60 R=RND(-TIME)
70 CLS
80 REM *
90 REM * Ввод имен
100 REM *
110 FOR I = 0 TO 9
120 PRINT I;
130 INPUT "Имя: ";A$(I)
140 NEXT
150 REM *
160 REM * Выбор имени случайным образом
170 REM *
180 R = INT(RND(1)*10)
190 N$ = A$(R)
200 PRINT
210 PRINT "Случайно выбранное имя: ";N$
220 END
```

Программа 6.2

```
10 REM *
20 REM * Бросание кубика
30 REM *
40 DIM V(6)
50 X=RND(-TIME)
60 INPUT "Сколько бросаний (до 100)";N
70 IF N<1 OR N>100 THEN 60
80 REM *
90 REM * Моделирование N бросаний
100 REM *
110 FOR I=1 TO N
120 T=INT(RND(1)*6+1):V(T)=V(T)+1
130 NEXT
140 REM *
150 REM * Вычисление частот
160 REM *
170 CLS
180 PRINT "Число бросаний:";
190 PRINT USING "###";N
200 PRINT
210 PRINT "Грань";TAB(8);"Число выпадений";
TAB(26);"Частота"
220 PRINT
```

```
230 FOR I=1 TO 6
240 PRINT TAB(1):I;
250 PRINT TAB(12): : PRINT USING "###":V( );
260 PRINT TAB(27);
270 PRINT USING "###.## %":(V(I)/N)*100
280 NEXT
```

```
Сколько бросаний (до 100) ? 85

Число бросаний : 85

Грань        Число выпадений        Частота

  1                19                22.35 %
  2                12                14.12 %
  3                13                15.29 %
  4                11                12.94 %
  5                11                12.94 %
  6                19                22.35 %
```

Рис. 6.2. Результаты работы программы 6.2

ВТОРИЧНОЕ ИНДЕКСИРОВАНИЕ

Содержимое одного массива можно использовать для индексирования другого. Этот прием называется ***вторичным индексированием***. Он, в частности, позволяет соединить строковый массив с числовым.

Индексный массив должен содержать значения индексов, служащих для обращения к отдельным массивам, в одном из которых хранятся числа, а в другом – строки символов. Программа 6.3 выполняет просмотр и сортировку при обработке платежной ведомости. Просмотр происходит обычным образом. Каждый элемент массива сравнивается с проверяемым словом для установления соответствия.

Программа 6.3

```
10 REM *
20 REM * Вторичное индексирование
30 REM *
40 CLEAR 1000
50 DIM IDX(10)
60 DIM NM$(10)
70 DIM SLY(10)
80 REM *
90 REM * Чтение массивов
100 REM *
110 FOR I=1 TO 10
```

```
120 IDX(I)=I
130 READ NM$(I),SLY(I)
140 NEXT
150 CLS
160 PRINT "1. Поиск"
170 PRINT "2. Сортировка данных"
180 PRINT
190 INPUT "Выберите номер ";A
200 IF A<1 OR A>2 THEN 190
210 ON A GOSUB 260,430
220 END
230 REM *
240 REM * Поиск
250 REM *
260 CLS
270 PRINT "Какое имя искать:"
280 INPUT S$
290 IF S$="" THEN 270
300 F=0
310 FOR I=1 TO 10
320 IF NM$(IDX(I))=S$ THEN S=SLY(IDX(I)) :F=-1
330 NEXT
340 IF NOT(F) THEN PRINT "Не найдено":GOTO 380
350 PRINT "Имя  : ";S$
360 PRINT "Оклад :";S
370 RETURN
380 BEEP
390 RETURN
400 REM *
410 REM * Сортировка
420 REM *
430 CLS
440 PRINT "1. A-Z"
450 PRINT "2. Z-A"
460 PRINT
470 INPUT "Выберите номер";A
480 IF A<0 OR A>2 THEN 470
490 CLS
500 PRINT "Идет сортировка.." : PRINT
510 REM *
520 REM * Сортировка по двум первым
530 REM * буквам имени
540 REM *
550 FOR I = 2 TO 10
560 FOR J = 10 TO 1 STEP -1
570 A$ = LEFT$(NM$(IDX(J-1)),2)
580 B$ = LEFT$(NM$(IDX(J)),2)
590 IF A=2 THEN SWAP A$,B$
600 FOR P=1 TO 2
```

```
610 X=ASC(MID$(A$,P,1))
620 REM •
630 REM • Замена строчных букв прописными
640 REM • при необходимости
650 REM •
660 IF X>90THEN MID$(B$,P,1)=CHR$(X-32)
670 X=ASC(MID$(B$,P,1))
680 IF X>90THEN MID$(A$,P,1)=CHR$(X-32)
690 NEXT P
700 IF A$>B$ THEN SWAP IDX(J-1),IDX(J)
710 NEXT J
720 NEXT I
730 FOR I=1 TO 10
740 PRINT NM$(IDX(I)),:PRINT USING "####
#";SLY(IDX(I))
750 NEXT I
760 RETURN
770 DATA "Tigger",5940
780 DATA "Bryant",7800
790 DATA "Cunliffe",15940
800 DATA "POTTS",10010
810 DATA "Major",8080
820 DATA "Tremayne",7100
830 DATA "Wirth",9500
840 DATA "Bartram",11340
850 DATA "Kernighan",6200
860 DATA "Sparks",7400
```

Сортировка выполняется путем последовательных обращений к массиву и анализа данных. При необходимости делаются перестановки элементов массива до тех пор, пока все данные не будут расположены в нужном порядке. Можно сортировать текстовую информацию как в прямом, так и в обратном алфавитном порядке (A–Z или Z–A).

Чтобы убедиться, что такие строки, как "BELL" и "BROWN", расположены правильно, необходимо сравнить по два их крайних левых символа. Перед сравнением все строчные буквы заменяются заглавными.

В данном случае сортируется именно массив индексов, и все происходящие изменения относятся к его содержимому. При обработке значительного числа связанных по смыслу массивов польза и удобство такого метода очевидны. В нашем примере при сортировке каждый раз требуется лишь однократный обмен значениями индексов вместо выполнения двух сортировок, затрагивающих и строковый, и числовой массивы.

ТАБЛИЦЫ

Списки также называют *одномерными массивами. Двумерные массивы (матрицы)* позволяют создавать и обрабатывать таблицы. Как и ранее, для описания массива используется оператор DIM. Например, оператор 10 DIM A(3,3) создаст таблицу размером 3*3. В программе 6.4 показано, как таблицы применяются в известной игре "морской бой". В

данном варианте играющий размещает шесть кораблей в пространстве 20*20. Затем компьютер случайным образом размещает свои корабли на собственном игровом поле. Оба игрока, человек и машина, поочередно пытаются угадать расположение кораблей противника. Игра заканчивается, если одному из партнеров удается полностью уничтожить флот другого.

Программа 6.4

```
10 REM *
20 REM * Морской бой
30 REM *
40 DIM CG(20,20),PG(20,20)
50 CS=0 : PS = 0
60 SCREEN 0:WIDTH 36:KEY OFF
70 PRINT TAB(2);
80 FOR J=65 TO 84
90 PRINT CHR$(J);
100 NEXT
110 PRINT
120 FOR J=65 TO 84
130 PRINT CHR$(J)
140 NEXT
150 REM *
160 REM * Заполнение игрового поля
170 REM *
180 LOCATE 0,24:PRINT "Разместите
свои корабли – буква: буква:";
190 FOR S=1 TO 6
200 LOCATE 22,24,1
210 A$=INPUT$(1)
220 IF A$<"A" OR A$>"T" THEN BEEP:GOTO 200
230 PRINT A$;
240 X=ASC(A$)-64
250 LOCATE 32,24,1
260 A$=INPUT$(1)
270 IF A$<"A" OR A$>"T" THEN BEEP:GOTO 200
280 PRINT A$;
290 Y=ASC(A$)-64
300 IF PG(X,Y)=0 THEN PG(X,Y)=1 ELSE
BEEP:GOTO 200
310 NEXT S
320 REM *
330 REM * Заполнение поля компьютера
340 REM *
350 FOR S = 1 TO 6
360 X=INT(RND(1)*20)+1
370 Y=INT(RND(1)*20)+1
380 IF CG(X,Y)=0 THEN CG(X,Y)=1 ELSE
```

```
GOTO 360
390 NEXT S
400 REM •
410 REM • Начало игры
420 REM •
430 LOCATE 28,0:PRINT "Ваш ход ":
440 LOCATE 22,24,0
450 A$=INPUT$(1)
460 IF A$<"A" OR A$>"T" THEN BEEP:GOTO 440
470 PRINT A$;
480 X=ASC(A$)-64
490 LOCATE 32,24,1
500 A$=INPUT$(1)
510 IF A$<"A" OR A$>"T" THEN BEEP:GOTO 490
520 PRINT A$;
530 Y=ASC(A$)-64
540 IF CG(X,Y)=99 THEN BEEP:GOTO 450
550 LOCATE X-1,Y:PRINT "X";
560 IF CG(X,Y)=1 THEN PS=PS-1:LOCATE X-1,Y:PRINT "S"
570 CG(X,Y)=99
580 IF PS=6 THEN CLS:PRINT "Вы выиграли!!!!":END
590 REM •
600 REM • Ход компьютера
610 REM •
620 LOCATE 28,0:PRINT "Компьютер"
630 FOR I=1 TO 500:NEXT I
640 X=INT(RND(1)*20)-1
650 Y=INT(RND(1)*20)-1
660 IF PG(X,Y)=99 THEN 640
670 IF PG(X,Y)=1 THEN CS=CS-1
680 IF CS=6 THEN CLS:PRINT "Я выиграл":END
690 PG(X,Y)=99
700 GOTO 430
```

Каждый ход отмечается в массиве: в соответствующее поле записывается число 99. Благодаря этому попытка хода на одну из уже "обстрелянных" позиций вызывает звуковой сигнал.

Можно работать с массивами и большего числа измерений. Так, оператор

```
10 DIM A(3,3,3)
```

описывает трехмерный массив, содержащий 27 элементов. Обычно массивы более чем трех измерений в практике почти не встречаются.

ВЕДЕНИЕ МАССИВОВ

Массив можно удалить из памяти по команде ERASE. Все пространство памяти, отведенное под массив с соответствующим именем, освобождается для других целей. Оператор

```
80 ERASE A!
```

уничтожает массив A!.

Единственное ограничение на размеры массива накладывается объемом памяти. Это особенно нужно учитывать при работе с многомерными и строковыми массивами.

РАБОТА С ФАЙЛАМИ

Кассетный накопитель позволяет хранить не только программы, но и необходимые для их работы данные. Данные располагаются на ленте последовательно. Их приходится считывать строго в том же порядке, в котором они были записаны на ленту. Такую форму хранения нельзя назвать ни гибкой, ни удобной, особенно в тех случаях, когда требуется часто обновлять и дополнять данные. (В то же время при долговременном хранении данных она полезна.)

Данные можно считывать с ленты, а также записывать на нее программным путем, если рассматривать кассетный накопитель как файл. Этим термином обычно обозначают набор данных. В MSX-Бейсике под файлами могут подразумеваться и периферийные устройства.

Перечислим основные процедуры, которые следует различать при обработке файлов:

1. Подготовка устройства к использованию в качестве файла. В MSX-Бейсике перед началом работы с файлом для него требуется отвести специальную область памяти – буфер. При этом файл считается открытым для дальнейшей обработки.
2. Ввод или вывод информации
3. Окончание работы с файлом – закрытие файла.

Если в роли файла выступает кассетный накопитель, он может быть открыт одним из двух способов: как входной (input) или выходной (output). Открытому файлу присваивается номер, который в дальнейшем задействуется при ссылках и обращениях. Одновременно в MSX-Бейсике может быть открыто до 16 файлов. Два файла открываются сразу, по умолчанию. Это число можно увеличить с помощью оператора MAXFILES. Оператор

```
10 MAXFILES = 4
```

резервирует буферную область для пяти файлов. Если MAXFILES=0 (что соответствует одному файлу), то можно выполнять лишь одну операцию – запись либо считывание.

Файлу, так же как и программе, которую требуется сохранить, можно присвоить имя. Имя представляет собой строку длиной до шести знаков, если файл размещается на кассете. Для именования файлов можно использовать символические имена устройств (см. гл. 1).

Существуют варианты оператора INPUT, в которые помимо команд ввода из файлов можно включать дополнительные команды вывода, аналогичные PRINT. Информация при этом записывается на кассету в формате ASCII.

Оператор CLOSE с номером файла прекращает все действия с данным файлом. Если он задан без аргументов, завершение операций ввода-вывода происходит во всех ранее открытых файлах. Кроме того, все открытые файлы закрывает и оператор END.

При закрытии кассетного файла на ленту записываются символы CTRL-Z. Все операторы ввода должны контролировать положение этих символов, отмечающих конец файла. Результат этой проверки можно сделать доступным программисту посредством функции EOF( ), которая при достижении маркера конца файла возвращает значение –1. Аргументом функции является номер проверяемого открытого файла.

Двигатель кассетного накопителя можно включать и выключать по команде MOTOR (включение – MOTOR ON, выключение – MOTOR OFF). Команда MOTOR без аргумента изменяет состояние двигателя: выключенный мотор включается и наоборот.

Простые примеры. В программе 6.5 вводится набор имен, который записывается на ленту, после чего эта информация может быть вновь считана и выведена на экран.

Программа 6.5

```
10 REM *
20 REM * Ввод-вывод файлов
30 REM *
40 CLS
50 PRINT "Файлы":PRINT
60 PRINT "Открытие файла.."
70 PRINT "Нажмите клавиши PLAY и RECORD"
80 OPEN "CAS:TEST" FOR OUTPUT AS #1
90 PRINT "Введите `*` для окончания"
100 PRINT
110 LINE INPUT "имя ":A$
120 IF A$="*" THEN GOTO 150
130 PRINT#1,A$
140 GOTO 110
150 PRINT "Закрытие файла"."
160 CLOSE#1
170 PRINT "Нажмите Rewind, а затем любую клавишу"
180 A$=INPUT$(1)
190 MOTOR
200 PRINT "Для прекращения перемотки нажмите клавишу"
210 A$=INPUT$(1)
220 MOTOR
230 PRINT "Нажмите PLAY и любую клавишу"
240 A$=INPUT$(1)
250 PRINT
260 OPEN "CAS:TEST" FOR INPUT AS #1
270 IF EOF(1) THEN GOTO 310
280 INPUT#1,A$
290 PRINT A$
```

```
300 GOTO 270
310 PRINT "Все данные считаны"
320 CLOSE
330 END
```

Нужно помнить, что оператор SAVE позволяет записывать программы на ленту в формате ASCII и впоследствии считывать внутрь программ подобно данным из файла. Аналогично можно записывать на ленту данные с целью последующей загрузки их оператором LOAD в качестве Бейсик-программы. Такие приемы используются в программах машинной графики, рассматриваемых в гл. 9. Программа 6.6 с помощью оператора LINE INPUT# выполняет простое считывание программы, записанной на ленту в кодах ASCII.

Программа 6.6

```
10 REM *
20 REM * Считывание и печать программы,
30 REM * записанной на ленту в формате
40 REM * ASCII (оператором SAVE)
50 REM *
60 SCREEN 0
70 INPUT "Имя программы: ";N$
80 OPEN "CAS:"+N$ FOR INPUT AS #1
90 IF EOF(1) THEN 130
100 LINE INPUT#1,A$
110 PRINT A$
120 GOTO 90
130 PRINT
140 PRINT "Конец программного файла"
150 CLOSE#1
160 END
```

Обработка записей. Создавая файл, содержащий имена людей и названия городов, где они живут, мы можем сохранить соответствующие записи на ленте . Каждая такая запись включает два поля: "имя" и "город". Записываемые на ленту данные разделяются запятыми, что позволяет позднее восстановить их всего одним оператором INPUT# . Практическая реализация этого приема демонстрируется программой 6.7.

Программа 6.7

```
10 REM *
20 REM * Вывод файла, состоящего из записей
30 REM *
40 CLS
50 REM *
60 REM * Подпрограмма вывода
70 REM *
```

```
80 PRINT "Открывается файл NAMES..."
90 OPEN "CAS:NAMES" FOR OUTPUT AS #1
100 PRINT "Имя: (* – для прекращения):"
110 INPUT N$
120 IF N$="*" THEN 200
130 PRINT "Город:"
140 REM *
150 REM * Подпрограмма ввода
160 REM *
170 INPUT C$
180 PRINT #1,N$:",":C$
190 GOTO 100
200 CLOSE#1
210 CLS
220 PRINT "Для поиска перемотайте ленту"
230 PRINT "Если готово, нажмите клавишу"
240 PRINT
250 A$=INPUT$(1)
260 OPEN "CAS:NAMES" FOR INPUT AS #1
270 IF EOF(1) THEN 130
280 INPUT#1,N$,C$
290 PRINT N$,C$
300 GOTO 270
310 PRINT "Конец файла"
320 CLOSE#1
330 END
```

Обработка файлов с использованием других устройств. Другие устройства, которые могут рассматриваться как файлы, относятся только к типу **output** (вывод). (Работа с графическим экраном обсуждается в гл. 8). Оператор OPEN для открытия таких файлов не должен содержать атрибутов ввода-вывода, требуется лишь номер файла. Принтер в качестве файла можно открыть таким образом:

```
10 OPEN "LPT:" AS # 1
```

Для устройств, являющихся только выводными, команда INPUT не имеет смысла. Программа 6.8 записывает данные на три различных устройства. Подобная техника применяется в генераторе программ, который рассматривается в гл. 9.

Программа 6.8

```
10 REM *
20 REM * Имена устройств в качестве файлов
30 REM *
40 CLS
50 MAXFILES=3
60 OPEN "CAS:" FOR OUTPUT AS #1
70 OPEN "LPT:" FOR OUTPUT AS #2
```

```
80 OPEN "CRT:" FOR OUTPUT AS #3
90 PRINT "1. Запись данных на кассету"
100 PRINT "2. Вывод данных на принтер"
110 PRINT "3. Вывод данных на экран"
120 PRINT
130 INPUT "Введите номер устройства";N
140 FOR I=1 TO 10
150 READ A$
160 PRINT #N,A$
170 NEXT
180 CLOSE
190 DATA Гайка,Болт,Гаечный ключ,Шуруп,Отвертка
200 DATA Молоток,Пила,Топор,Гвозди,Тиски
```

Использование файлов особенно полезно в тех случаях, когда данные хранятся значительное время, а также при ограниченном объеме памяти, отводимой для массивов. Последовательный характер записей на кассету делает этот способ неудобным для ведения адресной или телефонной записной книжки. Прикладные программы, очевидно, требуют значительных объемов памяти, что исключает применение массивов. Обычный хороший справочник адресов гораздо более удобен в эксплуатации, чем его кассетный эквивалент, и позволяет быстрее найти нужную информацию. Добавление в последовательно организованный справочник единственной записи влечет за собой просмотр и перезапись всего файла данных.

СВОДКА СЛУЖЕБНЫХ СЛОВ

Обозначения устройств

CAS: кассетный накопитель
LPT: принтер
CRT: текстовый экран

Операторы работы с массивами

DIM <имя переменной>(максимальный индекс[,<максимальный индекс>]...)[,<имя переменной>(...),...]
ERASE [<имя массива>,[<имя массива>]...]

Команды работы с файлами

OPEN "<имя устройства>[<имя файла>]" [FOR <INPUT|OUTPUT>
AS #<номер файла>]
CLOSE [#<номер файла>]
INPUT #<номер файла>,<имя переменной>[разделитель<имя переменной>]...
LINE INPUT #<номер файла>,<имя переменной>[разделитель <имя переменной>]...

PRINT #<номер файла>,[<выражение>]
PRINT #<номер файла>,USING<формат>;[<выражение>]
MAXFILES=<максимально допустимое количество одновременно открытых файлов>
EOF(<номер файла>)
MOTOR [<ON|OFF>]

Глава 7. ЗВУКОВЫЕ ВОЗМОЖНОСТИ MSX

Информация, содержащаяся в предыдущих главах, во всяком случае ее наиболее значительная часть, не привязана к какому-то конкретному диалекту Бейсика и имеет смысл для большинства из них. Что же касается таких разделов программирования, как работа со звуком или графикой, то соответствующие операторы Бейсика в различных реализациях выглядят по-разному. Возможности языка во многом определяются наличием специализированного оборудования. В данной и последующих главах эти устройства будут рассмотрены более детально.

ЗВУКОВЫЕ ЭФФЕКТЫ

Разумное применение звуковых эффектов значительно оживляет многие программы. В простейшем варианте звуковой сигнал позволяет убедиться в работоспособности программы. Звонок может привлечь внимание пользователя к ошибке ввода или подтвердить нажатие клавиши.

В современных компьютерных играх не редкость различные вспышки, взрывы — и все это на фоне немилосердного музыкального сопровождения. Как ни странно, микрокомпьютеры сами по себе трудно назвать хорошими музыкальными инструментами, поскольку диапазон и качество генерируемых ими звуковых сигналов сравнительно ограничены. И сочинять музыку гораздо легче с помощью обычных инструментов. Однако те, кто готов уделить изучению этого вопроса время и внимание, будут вознаграждены, так как MSX-Бейсик обладает широким спектром музыкальных возможностей.

Микросхема, обеспечивающая получение разнообразных звуков в системе MSX, представляет собой программируемый звуковой генератор AY–3–8912 фирмы General Instruments (сокращенно ПЗГ или PSG). Это устройство имеет двойное назначение, оно подключается к портам ввода-вывода для джойстика. Генератор может выдавать звуковые сигналы в диапазоне восьми октав с использованием трех независимых звуковых каналов ("голосов").

Помимо этого, к любому из звуковых каналов можно добавить "порцию" шума. MSX-Бейсик допускает два режима управления звуком: один из них явно ориентирован на музыку, второй более примитивен, зато генерируемые звуки намного разнообразнее.

КОМАНДА PLAY

Команда PLAY относится к музыкально-ориентированным элементам Бейсика. Отметим, что во всех примерах программ, приведенных в данном разделе, рекомендуется перед запуском программы выполнить команду BEEP. Последняя инициализирует звуковую микросхему, устанавливая ее регистры по умолчанию в первоначальное состояние.

Звуки, генерируемые по команде PLAY, определяются набором параметров, применяемых в виде символьных строк. Эти параметры в совокупности образуют ***музыкальный макроязык*** (ММЯ). Они, в частности, позволяют описать такие важные характеристики нотного текста, как высота тона, длительность звучания, темп, громкость. Элементы макроязыка описаны ниже.

Установка тона. Как уже отмечалось, в допустимом диапазоне содержатся восемь октав, или 96 звуков. Простейший способ получить тот или иной звук сводится к выполнению команды PLAY с соответствующим именем ноты. В ММЯ ноты обозначаются стандартным способом, принятым в музыкальной литературе: C (до), D (ре), E (ми), F (фа) и т.д. Диезы и бемоли (полутона, или, если угодно, черные клавиши рояля) обозначаются по-разному: C# или C+ обозначает до-диез, а B– обозначает ре-бемоль. Приведем пример, непосредственно демонстрирующий эту технику в действии.

```
PLAY "CC+DD+EFF+GG+AA+B"
```

Такую последовательность параметров можно оформить в виде строковой переменной и включить в программу, как показано ниже.

```
10 A$= "CC+DD+EFF+GG+AA+B"
20 PLAY A$
```

Отметим, что подобная запись допустима только для "правильных" нот. Поскольку в обычной музыкальной нотации не принято использовать си-диез или ми-диез, параметры B# и E# вызовут сообщение об ошибке.

До сих пор мы не выбирали октаву, в которой исполняется последовательность нот. По умолчанию устанавливается четвертая октава от C до B, т.е. от ноты до до ноты си включительно. Для изменения номера октавы служит параметр O, сопровождаемый номером октавы от 1 до 8 включительно, например O5 или O3. В следующем примере несложная последовательность звуков повторяется для трех разных октав.

```
PLAY "O1CDEFGO4CDEFGO8CDEFG"
```

В обычной последовательности ноты при звучании непрерывно сменяют друг друга. В то же время имеется возможность определить паузы — периоды молчания. Паузы, или "молчащие ноты", описываются параметром R и вводятся в строку так, как показано в следующем примере:

```
PLAY "O4CRO5CRO6C"
```

Наконец, высоту тона позволяет установить параметр N. Вместо имени ноты можно задавать ее номер от 0 до 96, причем 0 обозначает паузу, 1 эквивалентно "O1C", 2 – "O1C+" и т.д., например:

```
PLAY "N1N48N0N96"
```

Изменение длительности звучания и темпа. Во всех предыдущих примерах все ноты (и паузы) имели одну и ту же длительность. Чтобы изменять ее, необходимо вспомнить некоторые основные музыкальные термины и понятия. В ММЯ для описания длительности звучания используется стандартная музыкальная терминология, но обозначения несколько отличаются. Длительность описывается параметром L, за которым следует число от 1 до 64. По умолчанию в MSX-Бейсике принимается длительность L4, что соответствует четверти. На рис. 7.1 изображено соответствие между обозначениями ММЯ и общепринятыми символами, изображающими целую ноту, половину, четверть и т.д.

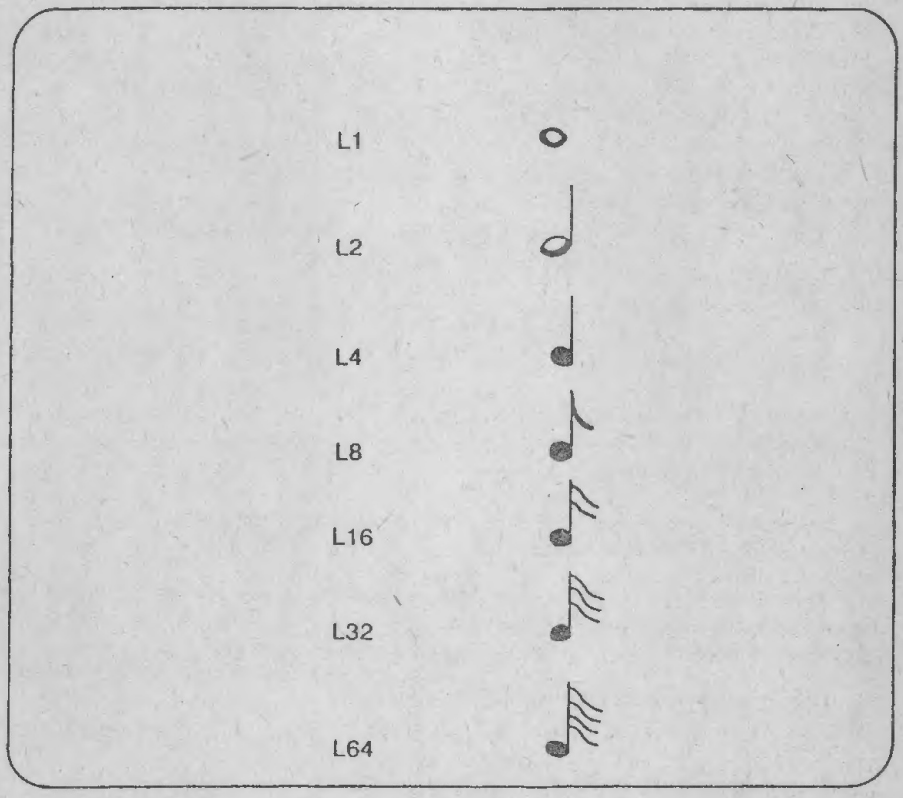

Рис. 7.1. Обозначения длительностей нот:
в ММЯ (слева) и традиционное (справа)

Это не единственный способ задания длительности нот и пауз. Можно применять его к каждой ноте индивидуально, сопровождая название ноты указанием ее длительности. Например, команда

```
PLAY "C4R4D4C8R16D16"
```

равнозначна такой:

```
PLAY "L4CRDL8CL16RD"
```

ММЯ позволяет также описывать "ноты с точкой", к длительности которых добавляется еще половина величины, установленной параметром L, например:

```
PLAY "CRC.RC.."
```

Таким образом, с помощью команды L можно устанавливать относительную длительность звучания нот. Например, целая нота, обозначаемая L1, звучит вдвое дольше половины (L2) и в четыре раза дольше четверти (L4). "Ноты с точкой" также можно описать в этих терминах.

Можно изменить и фактическую длительность сразу всех нот (не меняя их относительных длительностей), регулируя темп музыки. Соответствующий параметр обозначается буквой T и сопровождается числом от 32 до 255, равным числу четвертей, звучащих в течение минуты. Темп, как и другие параметры ММЯ, имеет значение, присваиваемое по умолчанию, в данном случае T120. Это означает, что за минуту можно исполнить 120 четвертей (или 30 целых нот). Чем больше значение T, тем быстрее темп. Программа 7.1 иллюстрирует эффект изменения темпа.

Программа 7.1

```
10 REM *
20 REM * Изменение темпа
30 REM *
40 A$ = "O4L8DF#ARAF#D"
50 FOR I = 1 TO 4
60 READ T$ : PLAY T$
70 PLAY A$
80 NEXT
90 DATA T32,T64,T128,T255
```

Изменение громкости и звуковые эффекты. Чтобы придать музыкальному фрагменту больше выразительности, можно регулировать громкость звучания. Наиболее примитивным средством ММЯ для изменения громкости служит команда V. Параметр громкости может принимать значения от 0 до 15, что соответствует диапазону от V0 (очень тихо) до V15 (громко до хрипоты). Мы не будем демонстрировать здесь эту возможность, так как того же эффекта можно добиться другими командами ММЯ. По умолчанию присваивается значение V8.

Значительно более интересная возможность ММЯ состоит в том, что он позволяет работать с восемью различными ***волновыми пакетами***, или ***формами волны***. Форма волны определяет параметры звука с точки зре-

ния изменения его громкости (амплитуды). Так, при одной форме волны звук, постепенно нарастающий по громкости, внезапно резко оборвется, другой волновой пакет соответствует обратной ситуации. Можно также получить с помощью компьютера трели, свист и другие интересные звуковые эффекты. Однако одновременно допускается пользоваться лишь одной из восьми форм, и разным звуковым каналам не могут соответствовать различные пакеты.

Для задания формы волны предусмотрены два параметра: S определяет форму волны, а M (*цикл модуляции*) – несущую частоту звукового генератора. На рис. 7.2 показаны допустимые формы волновых пакетов. Отметим, что, хотя параметр S может принимать значения от 0 до 15, некоторым из них соответствуют одинаковые пакеты. Причина этого поясняется в разделе "Команда SOUND". Выполнив программу 7.2, можно услышать особенности каждого из восьми пакетов.

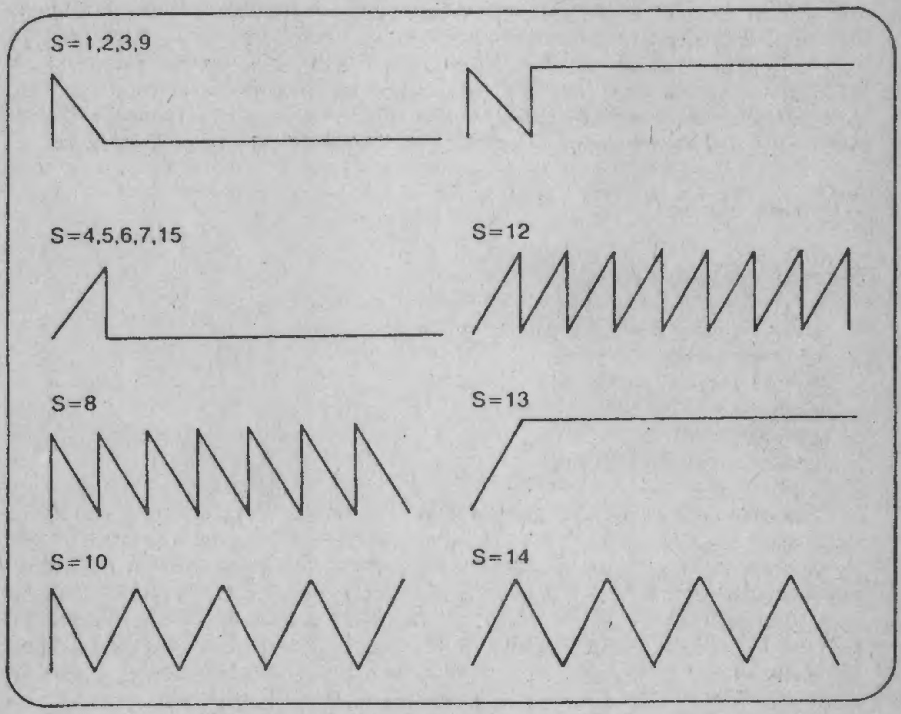

Рис. 7.2. Формы волновых пакетов

Программа 7.2

```
10 REM *
20 REM * Восемь волновых пакетов
30 REM *
40 CLS
50 A$ = "M500O4L4DF#ARAF#D"
60 FOR I = 1 TO 8
70 READ S$ : PRINT "Волновой пакет ";S$
80 PLAY S$
90 PLAY A$
100 FOR J = 1 TO 2000 : NEXT J
110 NEXT I
120 END
130 DATA S1,S4,S8,S10,S11,S12,S13,S14
```

Теперь рассмотрим, на что влияет изменение цикла модуляции. Работая с пакетом S8, мы увидим, как при трех разных значениях M генерируются совершенно различные способы звучания нот (программа 7.3). Чем меньше значение M, тем выше частота модуляции. Из трех полученных с помощью данной программы звуков наименьшей величине M соответствуют частые пульсации, в то время как наибольшей – звуки размеренных ударов. Продолжая экспериментировать с параметрами звуковых команд, можно обнаружить, что параметр V сбрасывает установленные значения S и M и меняет их на значения, принятые по умолчанию. Если перед командой S выполнить команду V, результат будет тем же самым. Другие варианты звучания вы можете получить с помощью программ 7.4 и 7.5.

Программа 7.3

```
10 REM *
20 REM * Изменение частоты
30 REM *
40 A$ = "S8O4L12CGAO5CDGA"
50 PLAY "M200" : PLAY A$
60 PLAY "M1000" : PLAY A$
70 PLAY "M5000" : PLAY A$
80 END
```

Программа 7.4

```
10 REM *
20 REM * Звуки (1)
30 REM *
40 PLAY "T250L4"
50 FOR I = 1 TO 4
60 PLAY "S14M400O4CAB"
70 PLAY "S8M600O6CO3CO2C"
```

```
80 PLAY "S14M400O4CO5BAO4CA"
90 PLAY "S8M600O4CO2CO3C"
100 NEXT I
110 END
```

Программа 7.5

```
10 REM •
20 REM • Звуки (2)
30 REM •
40 PLAY "T120L2"
50 PLAY "S13M9000O3F+RGACD"
60 PLAY "S13M9000O3F+RGACO2A"
70 FOR I = 1 TO 4
80 PLAY "S1M1500L64O5F+GACO4A"
90 PLAY "S11M900L64O3F+GACO2A"
100 NEXT I
110 END
```

Описанием команд формы волнового пакета и модуляции мы завершаем знакомство с основными элементами ММЯ. Приведенной информации вполне достаточно для программирования несложных звуковых эффектов, однако как команда PLAY, так и ММЯ имеют и дополнительные средства.

Использование переменных и подстрок. При описании модуляции, высоты тона и т.п. во всех командах ММЯ мы работали с постоянными параметрами. Новые интересные возможности открываются при включении в строки ММЯ переменных. Это достигается применением параметров, сопровождаемых знаком равенства, именем переменной и точкой с запятой. Например,

```
V = I;
```

(точка с запятой необходима во избежание ошибок). В программе 7.6 громкость звучания регулируется в цикле FOR...NEXT. Аналогичный прием можно применить ко всем ранее разобранным параметрам ММЯ.

Программа 7.6

```
10 REM •
20 REM • Изменение силы звука
30 REM •
40 FOR I = 15 TO 1 STEP -2
50 PLAY "V=I;T255L64O3DO2DO5CDEGFEO3ADECD"
60 NEXT I
70 END
```

Еще один вариант — включение в программу подстрок. Предположим, в музыкальном фрагменте какая-то фраза повторяется неоднократно. Вместо того чтобы всякий раз переписывать эту фразу, можно

обозначить ее как строковую переменную, а при необходимости вызывать как подстроку с помощью последней из команд ММЯ – X (строковая переменная). Она сопровождается именем строковой переменной и точкой с запятой, например XA$; (см. программу 7.7).

Программа 7.7

```
10 REM *
20 REM * Пример использования
40 REM * подстроки A$
50 REM *
60 A$ = "T180O3F32F# 32G4"
70 PLAY "T180XA$;XA$;F# 4F4D# 4C2.O2G2"
80 PLAY "RXA$;XA$;F# 4F4D# 4C2."
90 END
```

МНОГОГОЛОСИЕ И БУФЕРИЗАЦИЯ

В предыдущих музыкальных примерах был задействован лишь один звуковой канал (голос). Используя все три канала, можно создавать аккорды (при одновременном звучании трех голосов) или вести в отдельности мелодию и аккомпанемент. Синхронное звучание трех голосов описывается оператором PLAY с тремя строковыми выражениями, разделенными запятыми. Начнем с исполнения аккорда

```
PLAY "T120O4D4","T120O4F# ","T120OA4"
```

Это относительно простой пример, поскольку ноты во всех трех каналах имеют одинаковые длительности. При вводе более сложных последовательностей команд могут возникнуть трудности, связанные со способом исполнения команды PLAY. Интерпретатор Бейсика должен анализировать строки, относящиеся к каждому каналу. Если один из каналов содержит более длинный список команд ММЯ, чем другой, время интерпретации увеличивается, что может вызвать некоторую задержку. Таким образом, звучание разных каналов будет смещено во времени и безнадежно рассинхронизируется. С более короткими строками работать, конечно, проще, но это занимает много времени.

При записи музыки для трех голосов необходимо тщательно следить за синхронизацией. Прежде всего, часто полезно длительности звучания всех нот для каждого канала в команде PLAY сделать одинаковыми, т.е. для одного голоса работать с целой нотой, для другого – с двумя половинами, для третьего – с восемью восьмыми. Как сочинение музыки, так и ее переложение (транскрибирование) требует тщательного предварительного обдумывания перед записью на ММЯ.

Вы можете заметить, что приглашение "Ok" может появиться на экране в то время, когда исполнение музыки еще не закончилось. В MSX часть ОЗУ отводится для музыкального буфера. При трансляции очередной строки ММЯ команды звуковой микросхемы попадают в очередь в этом буфере и периодически исполняются. У такой системы есть свои

преимущества. В процессе работы вашей программы, выполняющей вывод информации на экран, математические вычисления или другие задания, в фоновом режиме может непрерывно звучать музыка. Неудобства проявляются при попытках обеспечить музыкальное сопровождение в определенные моменты выполнения программы, например при запросе ввода данных с клавиатуры или выдаче сообщений на экран.

Программа 7.8 наглядно демонстрирует описанный выше эффект очереди. Замысел состоял в том, чтобы последовательное исполнение десяти нот одновременно сопровождать индикацией на дисплее названия ноты. Как легко убедиться, названия нот появляются почти одновременно в виде списка, без всякой связи с их исполнением. Наиболее простое решение проблемы сводится к организации цикла временной задержки внутри основного цикла FOR. Период задержки необходимо согласовать с темпом и длительностью звучания нот.

Программа 7.8

```
10 REM *
20 REM * Рассогласование времени
30 REM *
40 CLS
50 PLAY "T255O5L16"
55 FOR I = 1 TO 10
60 READ A$
65 PRINT "Название ноты ";A$
70 PLAY A$
80 NEXT I
90 END
100 DATA C,A,E,F,G,B,D,G,E,C
```

MSX-Бейсик предоставляет программисту довольно ограниченные возможности контроля за тем, что происходит в музыкальной очереди. Единственная встроенная функция, предназначенная для обеспечения музыкального сопровождения, носит то же имя PLAY, хотя это может вызвать некоторую путаницу. Функция PLAY определяет наличие в очереди неисполненных нот и возвращает значение, указывающее, пуста очередь или нет. Функция PLAY может иметь один из четырех аргументов. Если в качестве аргумента задается число от 1 до 3, определяется состояние лишь канала с данным номером. При нулевом аргументе проверяются все каналы. Функция PLAY при любом значении аргумента возвращает значение –1 (истина), если очередь не пуста, и 0 (ложь) в противном случае.

С помощью этой функции можно управлять исполнением музыки в фоновом режиме. Периодически проверяя состояние музыкального канала, можно "засечь" момент, когда очередь опустеет, и добавить следующую порцию музыкальных данных. Если требуется обеспечить непрерывное звучание, необходимо тщательно подобрать частоту, с которой выполняется функция PLAY.

Фоновый режим исполнения музыки устанавливается программой 7.9, основное назначение которой — генерация случайных строк символов. Возможно, получаемое качество звучания далеко от совершенства, тем не менее оно дает представление о том, каких результатов вы можете добиться.

Программа 7.9

```
10 REM *
20 REM * Фоновая музыка
30 REM *
40 CLS : KEY OFF
50 R = RND(-TIME)
60 REM *
70 REM * Описание подстрок
80 REM *
90 A$ = "O5DF#GA"
100 B$ = "O5GBO6CD"
110 C$ = "O5AO6C#DE"
120 D$ = "XA$;XC$;XB$;XA$;XB$;XC$;"
130 PLAY "V10T120L16"
140 PLAY D$
150 IF NOT(PLAY(1)) THEN 140
160 REM *
170 REM * Генерация случайных строк
180 REM *
190 W$ = ""
200 FOR I = 1 TO 5
210 X=INT(RND(1)*26)
220 W$ = W$ + CHR$(65+X)
230 NEXT I
240 PRINT W$,
250 GOTO 150
```

Наконец, программа 7.10 демонстрирует исполнение музыки в три голоса. В ней используется большинство средств музыкального макроязыка, в том числе звуковые пакеты, переменные и подстроки.

Программа 7.10

```
10 REM *
20 REM * Пример трехголосия
30 REM *
40 REM * Определение подстрок
50 REM *
60 A$ = "O2G4F8C8R8D8C8D8"
70 B$ = "O6D4R8O4D4A8O5D8O6A8"
80 C$ = "O5B8G8D4A4O5C4"
90 D$ = "O5F16G16A16B16O6C16"
```

```
100 E$ = "O3G4F8C8R8D8C8D8"
110 F$ = "O5F16G16B16C16O5D4"
120 G$ = "O2D4R8D8R8D8R4F8F8R8F8R8F8R8F8G4R8"
130 H$ = "O3D4R8D8R8D8R4F8F8R8F8R8F8R8F8G4R8"
140 I$ = "O3G4R8O2G4E8O2G8O3E8"
150 J$ = "O2G4F8C8R8D8O4C8O3B8"
160 K$ = "G8R8G8R4G4R8G8R8G8R4"
170 REM *
180 REM * Аккорд
190 REM *
200 PLAY "O4T100LIS13M23000G","O5T100LIS
13M23000F","O3T100LIS13M23000D"
210 PLAY "G","B","O5D"
220 PLAY "E","G","O4A"
230 PLAY "O3E","O5G","O4B"
240 PLAY "O3D","O5A","O4F#"
250 PLAY"O5G","O4E","O3D"
260 PLAY "M30000O4D","M30000O5D","M30000O3D"
270 REM *
280 REM * Установка громкости и
290 REM * выключение модуляций
300 REM *
310 PLAY "V10T150","V10T150","V11T150"
320 FOR I = 4 TO 6
330 J = I+1
340 L$ = "L24O=I:DEFGO=J:CO=I:BAG"
350 PLAY L$,L$,L$
360 NEXT
370 FOR I = 1 TO 2
380 PLAY B$,B$,B$
390 NEXT I
400 FOR I = 1 TO 2
410 PLAY B$
420 PLAY A$,E$,B$
430 NEXT
440 FOR I = 1 TO 2
450 PLAY A$,B$,C$
460 PLAY A$,B$,D$
470 PLAY A$,B$,C$
480 PLAY J$,B$,F$
490 NEXT I
500 PLAY G$,H$,C$
510 PLAY K$,K$,F$
520 PLAY G$,H$,"XB$;XC$;"
530 PLAY K$,K$,F$
540 PLAY G$,H$,"XB$;XC$;"
550 PLAY "R4","R4","R4"
560 REM *
570 REM * Постепенное затухание
```

```
580 REM *
590 FOR I = 11 TO 0 STEP -2
600 PLAY "R","R","T255V=I:XF$;"
610 PLAY "R","R","XC$;"
620 NEXT I
630 END
```

КОМАНДА SOUND

Музыкальный макроязык является прекрасным инструментом, если нужно генерировать лишь звуковые сигналы, соответствующие стандартной (западной) музыкальной шкале. Для воспроизведения же звуковых эффектов, подобных шуму прибоя или сирене, требуется более точное управление звуковым генератором. Команда SOUND позволяет весьма детально описывать устанавливаемые частоты и формы волновых пакетов в дополнение к шумовым эффектам по любому индивидуальному каналу. В жизни часто бывает так, что чем большие вам предоставляются возможности, тем тщательнее и подробнее приходится изучать правила работы. Не составляет исключения в этом отношении и команда SOUND.

Формат команды и установка регистра. После обилия возможностей, предоставляемых командой PLAY, простота команды SOUND может показаться новичку несколько неожиданной. Формат этой команды:

SOUND <номер регистра ПЗГ>,<записываемое значение>

В MSX-Бейсике допускается запись значений в 14 регистров микросхемы. Существуют и другие регистры, но они связаны с работой джойстика и не участвуют в генерации звука. Содержимое всех 14 этих регистров (или некоторых из них) определяет характер получаемого звука. В табл. 7.1 описаны функции регистров и приведены предельно допустимые для каждого регистра значения записываемых величин.

Таблица 7.1

Номер регистра			Назначение или содержимое	Записываемые значения
0,	2,	4	Нижние 8 бит частоты голосов A,B,C	0–255
1,	3,	5	Верхние 4 бита частоты голосов A,B,C	0–16
		6	Регистр для генератора шума	0–31
		7	Управляющий регистр смешивания	0–63
8,	9,	10	Регистр управления амплитудой каналов A,B,C	0–16
		11	Нижние 8 бит управления периодом пакета	0–255
		12	Верхние 8 бит управления периодом пакета	0–255

Рассмотрим примеры использования перечисленных в таблице регистров. SOUND относится к числу тех команд Бейсика, с которыми интересно экспериментировать. Здесь уместно напомнить, что при утрате контроля за событиями всегда можно вернуться к начальному состоянию с помощью команды BEEP.

Регистр смешивания и выбора канала. Пожалуй, из всех регистров ПЗГ основным при работе является регистр 7. Его главное назначение – определять, какие каналы должны участвовать в образовании звука. Он также позволяет сопровождать звучание по одному или по всем каналам шумовыми эффектами. Под шумовыми эффектами мы подразумеваем шипящие или свистящие звуки, позволяющие имитировать работу паровой машины, шум прибоя и т.п. Проследить действие этого регистра удобнее всего, если рассматривать его как элемент двоичной памяти. Шесть младших двоичных разрядов связаны с управлением звуковыми каналами. Формат регистра показан на рис. 7.3.

В отличие от рассматривавшихся ранее двоичных значений, в этом случае нуль соответствует включению определенной позиции (истина), а единица – выключению (ложь). На рис. 7.3 нули в позициях с B0 по B2 указывают, что звук вырабатывается по всем трем каналам. Если же значения всех шести битов (B0 – B5) равны нулю, то по всем каналам генерируется и звук, и шум. Приведем несколько примеров двоичных чисел и отвечающих им состояний регистра смешивания:

&B00111000	(56)	По всем каналам – только звук.
&B00000111	(07)	По всем каналам – только шум.
&B00110110	(54)	Звук и шум по первому каналу.
&B00001100	(12)	Шум по третьему каналу, звук и шум по второму, только звук – по первому.

Установка частоты. Регистры с нулевого по пятый управляют значениями частот для всех трех звуковых каналов. К примеру, сочетание значений регистров 0 и 1 определяет частоту звука для первого канала. Все получаемые частоты являются делителями числа 1789800, которое выражает в герцах (колебаниях в секунду) значение частоты звукового генератора. Это составляет примерно половину тактовой частоты микропроцессора Z80, равной 3579545 Гц.

Для получения заданной частоты применяется несложная формула. Общее уравнение для звуковой платы выглядит так:

$$\frac{1789800}{16 * (\text{частота в Гц})} = 256 * (FT + CT).$$

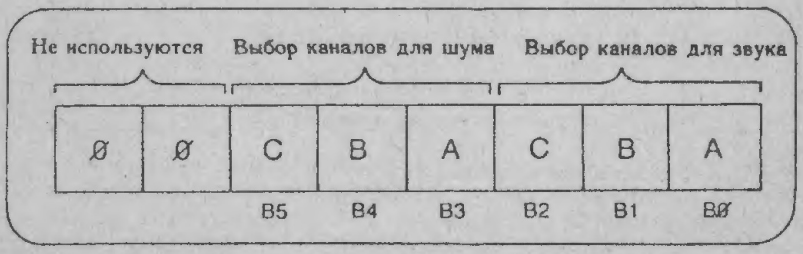

Рис. 7.3. Регистр смешивания и выбора канала (регистр 7)

Здесь FT – значение одного из регистров точной настройки тона (регистров 0, 2 или 4), а CT – значение парного ему регистра грубой настройки тона (регистров 1, 3 или 5). Составив на Бейсике программу для решения этого уравнения, можно вычислить значения регистров для любых заданных частот (см.программу 7.11).

Программа 7.11

```
10 REM *
20 REM * Исполнение ноты заданной частоты
30 REM *
40 SCREEN 0 : KEY OFF
50 CLS
60 PRINT
70 PRINT "Вычисление частоты"
80 PRINT
90 PRINT
100 INPUT "Введите частоту в Гц ";HZ
110 IF HZ<28 OR HZ>55932! THEN 50
120 PRINT
130 TMP = 1789800# / (16*HZ)
140 CT = INT(TMP/256)
150 FT = TMP MOD 256
160 PRINT "Частота";TAB(17);":";HZ;"Гц"
170 PRINT "Параметр точной настройки :";FT
180 PRINT "Параметр грубой настройки :";CT
190 PRINT
200 PRINT "Будете ли продолжать (Y-да,N-нет)?";
210 A$=INKEY$ : IF A$="" THEN 210
220 IF A$="Y" OR A$="y" THEN 50
230 END
```

Программа 7.12

```
10 REM *
20 REM * Определение частоты
30 REM * по значениям регистров
40 REM *
50 DEF FNFC(A,B)= INT((1789800# / ((256*A)+B))/16)
60 CLS
70 PRINT
80 PRINT "Вычисление частоты по значениям регистров"
90 PRINT
100 INPUT "Параметр точной настройки : ";FT
110 IF FT < 0 OR FT > 255 THEN BEEP:GOTO 100
120 INPUT "Параметр грубой настройки : ";CT
130 IF CT < 0 OR CT > 15 THEN BEEP:GOTO 120
140 REM *
```

```
150 REM * Исключение деления на нуль
160 REM *
170 IF CT=0 AND FT=0 THEN BEEP: GOTO 100
180 PRINT
190 PRINT "Частота ";CHR$(247);FNFC(CT,FT);" Гц"
200 PRINT
210 PRINT "Будете ли продолжать (Y-да,N-нет)?";
220 A$ = INKEY$ : IF A$ = "" THEN 220
230 IF A$ = "y" OR A$ = "Y" THEN 60
240 END
```

Программа 7.12 позволяет решить обратную задачу: по значениям пары регистров определить частоту в герцах.

В приложении 5 приведены значения регистров, необходимые для получения любых нот из допустимого диапазона ММЯ.

Изменение частоты шума. Регистр 6 управляет частотой генерируемого шума и позволяет изменять ее. Содержимое этого регистра определяется соотношением:

$$\frac{1789800}{16 * (\text{частота в Гц})} = \text{Значение регистра 6.}$$

Здесь в качестве значений регистра допускаются лишь числа от 0 до 31.

Изменение амплитуды и модуляционные эффекты. Регистры 8, 9, 10 регулируют амплитуду (громкость) звука соответственно для первого, второго и третьего каналов. При записи значений в эти регистры возникает тот же эффект, что и при работе команды V музыкального макроязыка. Однако при записи в один из трех регистров значения 16 появляется возможность работы с выбранной формой волнового пакета. Рис. 7.4 иллюстрирует двоичное представление регистра управления амплитудой.

Теперь легко понять причины "антагонизма" команд S и V. При выполнении команды S в соответствующий регистр управления амплитудой записывается число 16. При этом в разряд B4 попадает единица, а в младшие разряды — нули, следовательно, все установленные значения амплитуд сбрасываются. Параметр V находится в диапазоне от 0 до 15, поэтому значение B4 всегда будет сброшено.

Рис. 7.4. Регистр управления амплитудой

Выбор формы пакета осуществляется при записи требуемого значения в регистр 13. Значащими в нем являются лишь четыре младших разряда. Однако в ряде ситуаций значения двух младших разрядов не играют роли, поэтому при записи в регистр чисел, не превышающих семи, в результате может получаться одна и та же форма пакета. Числам, большим семи, однозначно соответствуют различные формы волновых пакетов.

Частота модуляции пакета определяется точно так же, как и частота звука. В этом случае в формулу нужно подставить значения регистров 11 и 12 и немного изменить программу 7.11.

Все пакеты можно подразделить на два типа: "непрерывные" и "скачкообразные". В первых, как следует из их названия, звук изменяется непрерывно. К ним относятся пакеты 8, 10, 12, 14. В "скачкообразных" пакетах происходит однократная модуляция. При этом всякий раз, когда требуется получить данный звук, необходимо записывать номер пакета в регистр 13.

ПРИМЕРЫ ЗВУКОВЫХ ПРОГРАММ

В этом разделе изложенный выше материал будет применен на практике. Здесь представлены шесть различных программ, реализующих звуковые эффекты. Все они снабжены пояснениями.

Программа 7.13. Сирена. Назначение программы состоит в генерации "завывающего" звука сирены, периодически равномерно нарастающего и затихающего. Для этого используется регистр, позволяющий изменять высоту исполняемой ноты. В данном примере высота круто нарастает, после чего падает до прежнего уровня.

Программа 7.13

```
10 REM
20 REM * Сирена
30 REM
40 SOUND 0,255
50 SOUND 1,0
60 SOUND 8,8
70 SOUND 7,&B00111110
80 REM *
90 REM * Нарастание высоты
100 REM *
110 FOR I = 252 TO 170 STEP -2
120 SOUND 0,I
130 NEXT
140 REM *
150 REM * Убывание высоты
160 REM *
170 FOR I = 170 TO 252 STEP 1
180 SOUND 0,I
190 NEXT
200 GOTO 110
```

Программа 7.14. Шум. Для одного канала генерируется шум, частота которого непрерывно меняется от высокого значения до низкого. Такой эффект достигается при изменении содержимого регистра 6.

Программа 7.14

```
10 REM *
20 REM * Шумовой эффект
30 REM *
40 SOUND 6,0
50 SOUND 7,&B00000111
60 REM *
70 REM * Убывание частоты шума
80 REM *
90 FOR I = 0 TO 31
100 SOUND 8,10 : SOUND 9,10 : SOUND 10,10
110 SOUND 6,I
120 FOR J = 1 TO 50 : NEXT J
130 SOUND 8,0 : SOUND 9,0 : SOUND 10,0
140 FOR J = 1 TO 25 : NEXT J
150 NEXT
160 GOTO 90
```

Программа 7.15. Звуковые эффекты. Здесь звучание основного тона сопровождается эффектом модуляции. При нажатии клавиши регистр 12 вызывает рост периода волнового пакета, а частота звука (регистр 1) изменяется случайным образом. Этот пример хорошо демонстрирует широчайший диапазон генерируемых звуков.

Программа 7.15

```
10 REM *
20 REM * Шумовые эффекты ("тема пришельцев")
30 REM *
40 SOUND 0,252
50 SOUND 1,0
60 SOUND 8,16
70 SOUND 11,200
80 SOUND 12,2
90 SOUND 13,10
100 SOUND 7,&B00111110
110 PRINT "Нажмите любую клавишу.."
120 A$ = INPUT$(1)
130 REM *
140 REM * Эффект модуляции: изменение
150 REM * частоты случайным образом
160 REM *
170 SOUND 12,1
180 SOUND 0,RND(1)*255
190 GOTO 180
```

Программа 7.16. "Перевернутые" звуки. Выбирается другая форма пакета, при которой амплитуда медленно растет, а затем резко падает. Используются все три звуковых канала, причем частота одного из них слегка "расстроена".

Программа 7.16

```
10 REM *
20 REM * "Перевернутые" звуки
30 REM *
40 R = RND(-TIME)
50 FOR I = 0 TO 13
60 READ A : SOUND I,A
70 NEXT I
80 FOR I=1 TO 1000:NEXT
90 X=RND(1)*250
100 IF X<30 THEN 90
110 Y=RND(1)*10+1
120 SOUND 0,X : SOUND 1,Y
130 SOUND 2,X : SOUND 3,Y
140 SOUND 4,X+5 : SOUND 5,Y
150 SOUND 13,13
160 GOTO 80
170 DATA 62,2,60,2,60,2,0,56,16,16,16,
120,30,13
```

Программа 7.17. Ударные эффекты. Звуки барабана можно имитировать, создавая шумовой эффект с помощью "скачкообразного" пакета. Для имитации ударов турецкого барабана используется тот же пакет, но на этот раз модулируется не шум, а звуковой тон. В зависимости от нажатой клавиши программа выдает то или иное звучание. Регистр 7 позволяет выделить получаемый звук.

Программа 7.17

```
10 REM *
20 REM * Звуки барабана
30 REM *
40 SCREEN 0,0,0
50 FOR I = 0 TO 12
60 READ S
70 SOUND I,S
80 NEXT I
90 PRINT "Нажмите `Б` для барабана"
100 PRINT "Нажмите `Т` для турецкого барабана"
110 A$ = INPUT$(1)
120 IF A$ = "Т" THEN SOUND 7,&B00111000:SOUND 13,1
130 IF A$ = "Б" THEN SOUND 7,&B00000111:SOUND 13,1
140 GOTO 110
150 DATA 140,7,140,7,140,7,15,255,16,16,16,190,7
```

Программа 7.18. Бой часов. Заключительная программа этого раздела воспроизводит бой часов. Звук генерируется с помощью набора трех частот, настроенных с некоторым рассогласованием. "Скачкообразный" пакет создает впечатление удара в колокол, а затухание звука достигается за счет длинного модуляционного цикла. При изменении частоты тона можно добиться звучания, напоминающего колокольный звон, звучание треугольника (музыкальный инструмент) или даже звук удара по металлической трубе.

Программа 7.18

```
10 REM •
20 REM • Бой часов
30 REM • Настройка звукогенератора
40 REM •
50 FOR I = 0 TO 12
60 READ S: SOUND I,S
70 NEXT I
80 WIDTH 40 : CLS : KEY OFF
90 PRINT "Часы бьют" : PRINT
100 SOUND 7,56
110 REM •
120 REM • Цикл боя часов
130 REM •
140 FOR I = 0 TO 12
150 SOUND 13,0
160 PRINT I; TAB(1)
170 FOR J = 1 TO 1200 :'NEXT J
180 NEXT I
190 PRINT : PRINT
200 PRINT TAB(1): "12 часов" : PRINT " и все отлично!"
210 END
220 DATA 229,0,97,0,115,0,0,63,16,16,16,190,120
```

СВОДКА СЛУЖЕБНЫХ СЛОВ

BEEP
SOUND ⟨номер регистра⟩,⟨записываемое значение⟩
PLAY ⟨строка⟩[,⟨строка⟩[,⟨строка⟩]]
PLAY (⟨музыкальный канал⟩)

Параметры команды PLAY

C,D,E,F,G,A,B	Исполнение ноты по названию
N⟨n⟩	Исполнение ноты с номером ⟨n⟩
R	Исполнение паузы
O	Установка текущей октавы

Дополнительные обозначения нот

+ или #	Исполнение диеза
–	Исполнение бемоля
.	Исполнение "ноты с точкой"
<n>	Исполнение ноты длительности <n>

Параметры формы звуковой волны

S<n>	Установка пакета
M<n>	Установка частоты модуляции

Прочие обозначения

V<n>	Установка амплитуды звука
L<n>	Установка длительности ноты
T<n>	Установка темпа
X<строковая переменная>	Выполнение музыкальной подпрограммы

Глава 8. ВВЕДЕНИЕ В ГРАФИКУ

MSX-Бейсик предоставляет программисту богатый набор функций и команд для создания данных в графической форме и выполнения над ними действий. Этот инструментарий можно применять для различных целей. Охарактеризуем кратко некоторые из областей его применения.

Графика. При работе с большими информационными массивами зачастую для понимания закономерностей в их структуре вполне достаточно качественных, а не количественных оценок. Данные можно представить в таких графических формах, как гистограммы (столбиковые диаграммы), круговые (секторные) диаграммы или пиктограммы. Благодаря своей способности обеспечивать быстрое сопровождение математических вычислений графическими образами компьютер становится незаменимым помощником исследователя, занимающегося численным анализом.

Интерфейс с пользователем. Привлекая символы графики, пользователь может упростить свою работу по программе и сделать ее гораздо более удобной. Вполне возможно, что наглядное изображение шкафа, полок и папок с документами позволит легче сориентироваться, чем понятие "реляционная база данных". В современных профессиональных компьютерах эта техника получает широкое распространение. Хотя все они являются 16-разрядными, многие идеи в их программном обеспечении заимствованы из MSX-Бейсика.

Игры. Разработка сценариев игр типа "Догони и уничтожь!" представляет, очевидно, наиболее широкое поле приложения компьютерной графики как таковой. Графические образы постоянно изменяются, что можно наблюдать в виде любопытных эффектов в играх, где окружающий мир выглядит трехмерным. Многим хорошо знакома та виртуозность, с которой на экране изображается полет космических ракет и снарядов, а также действия разнообразных чудовищ.

В этой и следующей главах мы разберем некоторые приемы работы с графикой.

ГРАФИЧЕСКИЕ ЭКРАНЫ

В MSX-Бейсике рассматриваются графические экраны двух типов, для которых приняты те же правила описания координат, что и для текстового экрана. В частности, точка (0,0) обозначает левый верхний угол экрана. Экраны обоих типов имеют размеры 256 * 192, что соответствует 49152 адресуемым точкам, но различаются по степени разрешения. Разрешающая способность экрана определяет размеры пиксела — минимального элемента изображения.

При низкой разрешающей способности экрана пиксел имеет размер 4 * 4 точки. Таким образом, на экране одновременно можно изобразить не более 64 * 48 отдельных пикселов. Это означает, что обращение к точке (128,96) даст такой же эффект, как и для точки (130,98).

На экране с высоким разрешением число пикселов равно числу адресуемых точек экрана. В большинстве рассматриваемых здесь примеров программ используется второй режим.

Формат экрана устанавливается по команде SCREEN. Для экрана с высоким разрешением нужно указывать SCREEN 2, с низким — SCREEN 3. Оба графических режима неустойчивы в отличие от текстового, где установленный формат сохраняется вплоть до отмены его следующей командой SCREEN. Если задать SCREEN 2 в виде непосредственной команды, на экране будет видна лишь яркая вспышка, сопровождаемая приглашением "Ok". Чтобы сохранился графический режим, в программе требуется бесконечный цикл, препятствующий возврату к текстовому формату.

Обращение к точкам. В Бейсике существует команда PSET, устанавливающая цвет определенного пиксела на экране. Координаты описываемой точки можно указывать двумя способами.

* Абсолютные координаты задаются непосредственным указанием точки на экране. Такой способ обращения к точкам наиболее распространен. Команда имеет следующий синтаксис:

PSET (<координата X>,<координата Y>)[<цвет>]

Если в качестве абсолютных координат взять отрицательные числа, то фактически присваиваются нулевые значения. Так, PSET(–45,20) означает то же самое, что и PSET(0,20).

* Относительные координаты задаются с помощью слова STEP. Координаты при этом определяют положение относительно последней адресованной графической точки. Величина смещения может, таким образом, быть как положительным, так и отрицательным числом. В данной форме оператор имеет вид:

PSET STEP (<сдвиг по X>,<сдвиг по Y>)[,цвет]

Эти два способа описания координат применяются во всех графических командах MSX.

В качестве координат можно выбирать любые целые числа, однако наблюдать действие оператора можно лишь в тех случаях, когда X лежит в диапазоне от 0 до 255, а Y — в диапазоне от 0 до 191. Действительные числа округляются до ближайшего целого.

Как следует из описания формата оператора PSET, параметр "цвет" не является обязательным. Если он опущен, отмеченная точка окрашивается в цвет установленного по умолчанию основного изображения. Это соглашение справедливо и для всех прочих графических команд, исключая PRESET. Оператор PRESET подчиняется тем же синтаксическим правилам, что и PSET, однако в отсутствие параметра "цвет" точка окрашивается в текущий цвет фона. Действие описанных операторов иллюстрируется программами 8.1, 8.2, 8.3 и рис. 8.1.

Программа 8.1

```
10 REM *
20 REM * Операторы PSET и PRESET
30 REM *
40 COLOR 15,1,1
50 SCREEN 2
60 REM *
70 REM * Белые точки
80 REM *
90 FOR X = 16 TO 240 STEP 4
100 FOR Y = 16 TO 176 STEP 4
110 PSET(X,Y)
120 NEXT Y
130 NEXT X
140 REM *
150 REM * PRESET удаляет точки
160 REM *
170 FOR Y = 176 TO 16 STEP -4
180 FOR X = 240 TO 16 STEP -4
190 PRESET(X,Y)
200 NEXT X
210 NEXT Y
220 REM *
230 REM * Поддержка графического режима
240 REM *
250 GOTO 250
```

Программа 8.2

```
10 REM *
20 REM * PSET в относительных координатах
30 REM *
40 COLOR 15,4,4
50 SCREEN 2
60 REM *
70 REM * Установка начальной точки
80 REM *
90 PRESET (0,0)
```

```
100 REM *
110 REM * Изображение прямоугольников
120 REM * в относительных координатах
130 REM *
140 FOR I=1 TO 255:PSET STEP(1,0):NEXT
150 FOR I=1 TO 191:PSET STEP(0,1):NEXT
160 FOR I=1 TO 255:PSET STEP(-1,0):NEXT
170 FOR I=1 TO 191:PSET STEP(0,-1):NEXT
180 REM *
190 REM * Смена начальной точки
200 REM *
210 PRESET (60,30)
220 FOR I = 1 TO 4
230 FOR J = 50 TO 10 STEP -10
240 FOR K = 1 TO J
250 PSET STEP(1,0)
260 NEXT K
270 FOR K = 1 TO J
280 PSET STEP(0,1)
290 NEXT K
300 FOR K = 1 TO J
310 PSET STEP(-1,0)
320 NEXT K
330 FOR K = 1 TO J
340 PSET STEP(0,-1)
350 NEXT K
360 PRESET STEP (5,5)
370 NEXT J
380 NEXT I
390 GOTO 390
```

Программа 8.3

```
10 REM *
20 REM * PSET при низком разрешении.
30 REM * Мозаика из случайных цветов
40 REM *
50 COLOR ,,15
60 SCREEN 3
70 FOR I = 0 TO 255 STEP 4
80 FOR J = 0 TO 191 STEP 4
90 PSET(I,J),RND(1)*16
100 NEXT J
110 NEXT I
120 GOTO 120
```

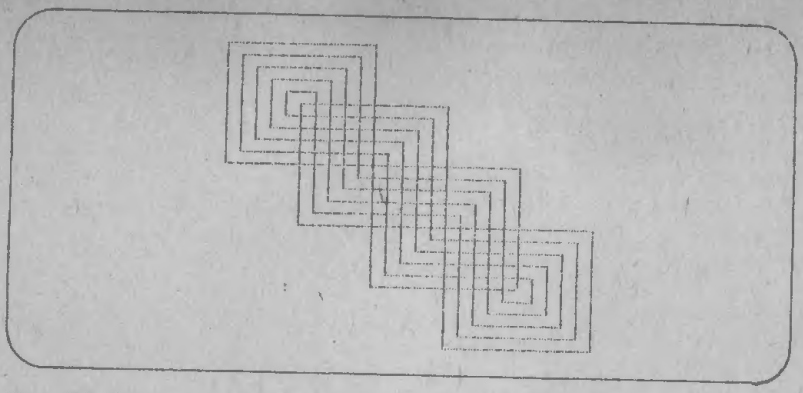

Рис. 8.1. Результат выполнения программы 8.2

Изображение прямых и прямоугольников. Среди команд MSX-Бейсика имеется команда, которая позволяет быстро изображать прямые линии и прямоугольники. Формат ее таков:

LINE [<указатель координат>]–<указатель координат>
[,<цвет>][,<B|BF>]

Два указателя координат отмечают начальную и конечную точки изображаемой линии. Обе эти точки можно задавать в относительной форме, т.е. координатам может предшествовать служебное слово STEP. При отсутствии первой пары координат за начальную точку линии берется текущая. Заметим, что символ "–" должен присутствовать всегда, например:

LINE -(128,96)

Чтобы получить прямоугольник, в конце оператора следует использовать параметр B или BF. В то время как B соответствует простому контуру прямоугольника, BF означает, что прямоугольник должен быть закрашен. В качестве указателей координат в этом случае берутся диагонально противоположные точки прямоугольника. Все возможные разновидности оператора LINE приведены в программах 8.4 и 8.5, а на рис. 8.2 представлены результаты его выполнения.

Программа 8.4

```
10 REM *
20 REM * Оператор LINE
30 REM *
40 COLOR 15,4,4
50 SCREEN 2
60 REM *
70 REM * Изображение прямых
80 REM *
```

```
90 FOR I = 120 TO 170 STEP 5
100 LINE (5,I)–(I,I)
110 NEXT I
120 REM •
130 REM • Изображение прямоугольников
140 REM •
150 FOR I = 10 TO 60 STEP 10
160 LINE (I,I)-(I+50,I+50),,B
170 NEXT
180 REM •
190 REM • Окрашенные прямоугольники
200 REM •
210 FOR I = 10 TO 125 STEP 35
220 LINE (170,I)-(220,I+25),,BF
230 NEXT
240 GOTO 240
```

Программа 8.5

```
10 REM •
20 REM • LINE с относительными координатами
30 REM •
40 COLOR 1,15,15
50 SCREEN 2
60 PSET(170,60)
70 FOR I = 5 TO 55 STEP 5
80 LINE –STEP(I,0)
90 LINE –STEP(I,I)
100 LINE –STEP(0,I)
110 LINE –STEP(–I,I)
120 LINE –STEP(–I,0)
130 LINE –STEP(–I,–I)
140 LINE –STEP(0,–I)
150 LINE –STEP(I,–I)
160 PRESET STEP(–7,–5)
170 NEXT
180 FOR I = 1 TO 500 : NEXT I
190 CLS
200 PRESET (70,30)
210 FOR I = 1 TO 30
220 LINE STEP(–45,2)–STEP(50,0)
230 NEXT
240 FOR I = 1 TO 30
250 LINE STEP(–55,2)–STEP(50,0)
260 NEXT
270 GOTO 270
```

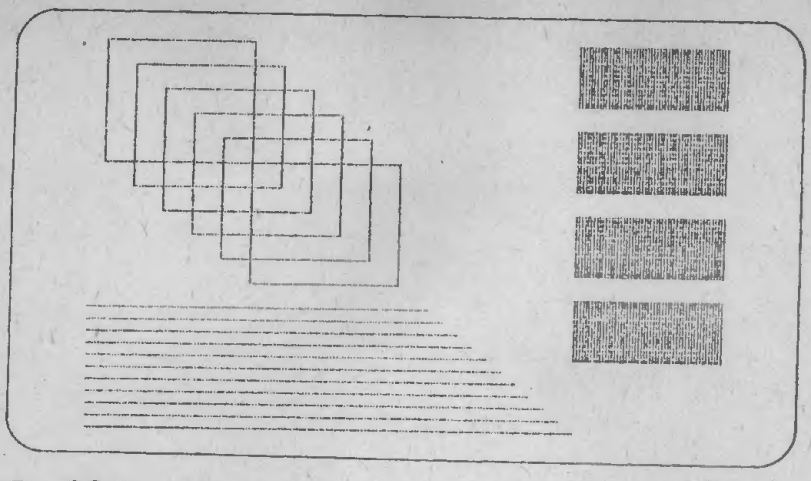

Рис. 8.2. Результат выполнения программы 8.4

Изображение эллипсов и дуг. Команда CIRCLE, пожалуй, является самой сложной в MSX-Бейсике. Она имеет следующий формат:

CIRCLE <указатель координат>,<радиус>[,<цвет>]
[,<начальный угол>][,<конечный угол>][,<овал>]

Указатель координат обозначает положение центра эллипса или дуги. С помощью рассматриваемой команды, даже в ее простейшем варианте, мы сможем получить различные эллипсы (программа 8.6).

Программа 8.6

```
10 REM *
20 REM * Изображение эллипсов
30 REM *
40 T=RND(-TIME)
50 COLOR 15,1,1
60 SCREEN 2
70 FOR I=1 TO 10
80 X = RND(1) * 224 + 16
90 Y = RND(1) * 160 + 16
100 FOR R=5 TO 20 STEP 2
110 CIRCLE(X,Y),R
120 NEXT R
130 NEXT I
140 GOTO 140
```

Радиус задается в виде целого числа. Если эллипс частично или полностью выходит за границы экрана, это не является ошибкой. Помимо полных эллипсов, команда CIRCLE позволяет изображать их

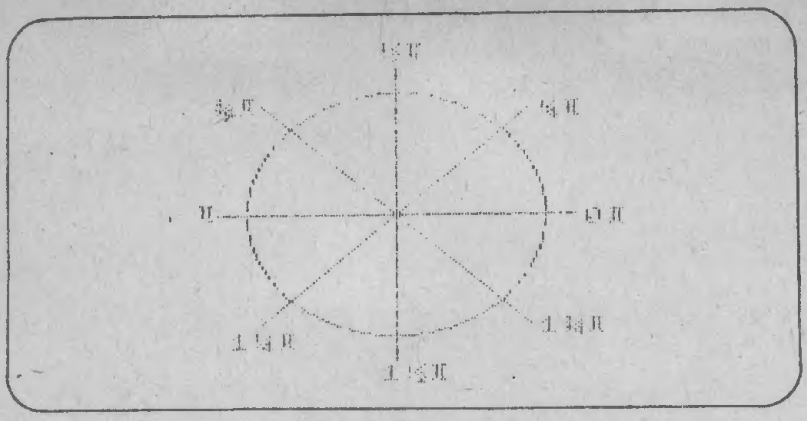

Рис. 8.3. К определению параметров оператора CIRCLE

фрагменты – дуги. Если в нее включить параметры "начальный угол" и "конечный угол", то их вполне достаточно для описания начала и конца дуги. Указанные параметры нужно вводить в радианах, поэтому их значения должны находиться между нулем и 2*PI. На рис. 8.3 показано соответствие между углами, измеряемыми в градусах и радианах. Его следует иметь в виду при работе с командой CIRCLE в графическом режиме.

Программа 8.7 показывает, что происходит при изменении значений начального и конечного углов. Если перед значением начального либо конечного угла стоит знак "минус", то линия будет изображаться, начиная с центральной точки. Пользуясь таким приемом, можно рисовать

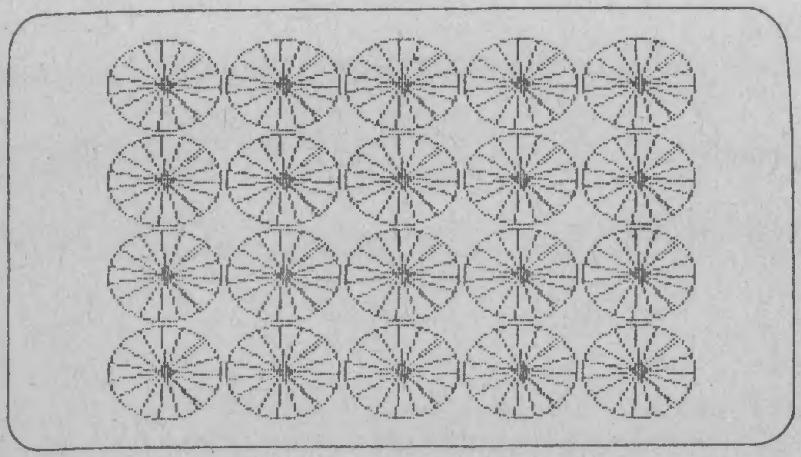

Рис. 8.4. "Колеса", изображенные с помощью программы 8.8

"колеса" и круговые диаграммы (см. рис. 8.4, который получен как результат работы программы 8.8).

Программа 8.7

```
10 REM •
20 REM • Начальные и конечные углы
30 REM •
40 COLOR 15,1,1
50 SCREEN 2
60 REM •
70 REM • Изменение начального угла
80 REM •
90 R = 96
100 FOR SA = 0 TO 6.28 STEP .4188
110 CIRCLE(128,96),R,15,SA,0
120 R= R-6
130 NEXT SA
140 FOR J = 1 TO 300 : NEXT J
150 CLS
160 REM •
170 REM • Изменение конечного угла
180 REM •
190 R = 96
200 FOR EA = 0 TO 6.28 STEP .4188
210 CIRCLE(128,96),R,15,0,EA
220 R= R-6
230 NEXT EA
240 GOTO 240
```

Программа 8.8

```
10 REM •
20 REM • Изображение радиусов
30 REM •
40 COLOR 15,1,1
50 C=5
60 SCREEN 2
70 FOR Y = 40 TO 160 STEP 40
80 FOR X=40 TO 216 STEP 40
90 FOR EA = 0 TO 6.29 STEP.4188
100 CIRCLE(X,Y),19,C,0,-EA
110 NEXT EA
120 NEXT X
130 C=C+2
140 NEXT Y
150 GOTO 150
```

И наконец, параметр "овал". Он обозначает соотношение осей, т.е. высоты эллипса и его ширины. Задавая значения, меньшие единицы, мы сможем строить эллипсы, сжатые по вертикали, в то время как значения, большие единицы, дадут нам вытянутые эллипсы (см. программу 8.9 и рис. 8.5). Если же положить "овал" равным единице, то, казалось бы, можно ожидать появления на экране правильной окружности. Однако в силу особенностей способа получения телевизионного изображения этого не происходит. Окружность, которая выглядит правильной, соответствует значению параметра 256/192 (или приблизительно 1.33).

Программа 8.9

```
10 REM *
20 REM * Изменение параметра "овал"
30 REM *
40 COLOR 15,1,1
50 SCREEN 2
60 FOR I=80 TO 24 STEP -4
70 CIRCLE(128,96),96,,,,I0/I
80 CIRCLE(128,96),96,,,,I/10
90 NEXT
100 GOTO 100
```

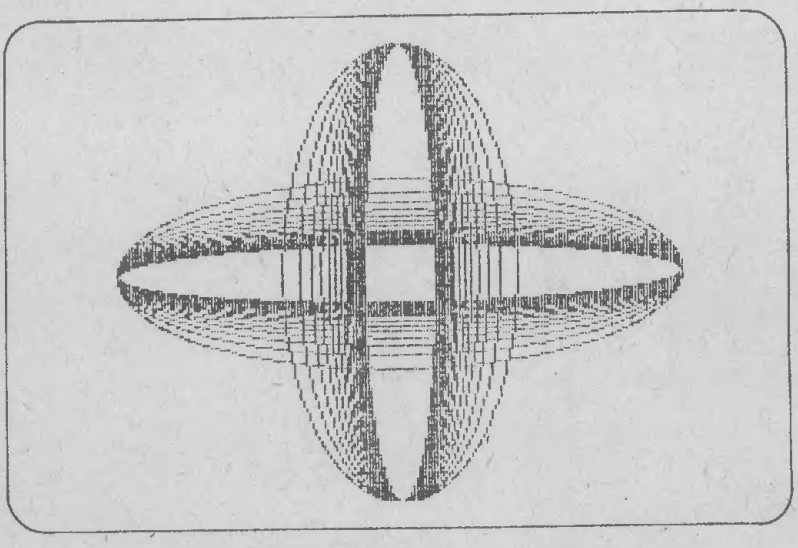

Рис. 8.5. Результат изменения параметра "овал" (программа 8.9)

ИСПОЛЬЗОВАНИЕ ЦВЕТА

В демонстрационных программах, рассматривавшихся до сих пор, использовались лишь два цвета. Здесь же мы познакомим вас со способами задания цвета основного изображения, фона и границы, а также с правилами, позволяющими управлять палитрой экрана.

Установка значений по умолчанию. Если в любой графической команде опустить параметр, регулирующий цвет, он будет установлен по умолчанию, причем берется то значение, которое фигурировало в последней из встретившихся команд COLOR (см. программу 8.10).

Программа 8.10

```
10 REM *
20 REM * Изображение прямоугольников
30 REM * с изменением начального цвета
40 REM *
50 SCREEN 2
60 C=0
70 FOR J = 0 TO 184 STEP 8
80 COLOR C
90 LINE (1,J)-STEP(256,8),,BF
100 C = C+1
110 IF C>15 THEN C = 1
120 NEXT J
130 GOTO 130
```

Отметим, что, хотя по команде COLOR изменяются устанавливаемые по умолчанию цвета изображения и фона, точки, уже имеющиеся на экране, свой цвет сохраняют. Сделать видимым изменение цвета фона можно с помощью комбинации команд COLOR и CLS. Правда, при этом весь экран окрашивается в цвет фона, а его содержимое делается невидимым. Команда COLOR позволяет, однако, сразу заметить изменение цвета границы, что иллюстрируется программой 8.11.

Программа 8.11

```
10 REM *
20 REM * Изменение цвета границы
30 REM *
40 COLOR 15,1,1
50 SCREEN 2
60 LINE (40,40)-(216,160),4,BF
70 FOR I=1 TO 15
80 COLOR ,,I
90 FOR J=1 TO 100:NEXT J
100 NEXT I
110 GOTO 70
```

Правило установки цвета при высоком разрешении. Если речь идет об использовании цвета, экран с высоким разрешением можно разбить на 6144 блока размерами 8 * 1 пикселов каждый. В частности, пикселы с координатами от (0,0) до (7,0) образуют один блок, с координатами от (8,0) до (15,0) — следующий и т.д. "Правило установки цвета" состоит в том, что в любом блоке допускается не более двух цветов. Программа 8.12 показывает, к каким последствиям приводит нарушение данного правила. Первоначально блок, имеющий координаты с (128,96) по (135,96), содержит лишь красный и белый цвета. Когда в этот блок, в позицию (129,96), помещается черный пиксел, все его красные пикселы меняют цвет на черный. Причина такого изменения цвета объясняется в гл.9, где рассматривается работа процессора видеодисплея. Лучший способ избежать подобных неприятностей — перед составлением программы тщательно разработать схему раскраски экрана.

Программа 8.12

```
10 REM *
20 REM * Нарушение "правила цвета"
30 REM *
40 COLOR 15,15,15
50 SCREEN 2
60 LINE (124,92)-STEP(8,8),8,BF
70 FOR I = 1 TO 500 : NEXT I
80 BEEP
90 PSET (129,96),1
100 GOTO 100
```

Экран с низким разрешением иногда называют многоцветным. Поскольку размер каждого пиксела составляет в этом варианте 4 * 4 точки, в блоке шириной в восемь точек просто не может содержаться более двух цветов, поэтому отмеченная выше проблема здесь не возникает.

РАСКРАШИВАНИЕ ОБЛАСТЕЙ ЭКРАНА

Область на экране можно "залить" определенным цветом с помощью оператора PSET, но это происходит слишком сложно и медленно. В MSX-Бейсике есть команда PAINT, которая позволяет раскрасить любую конфигурацию за вполне приемлемое время, но имеет некоторые синтаксические особенности в зависимости от типа используемого графического экрана.

Раскрашивание при высоком разрешении. Оператор PAINT в режиме SCREEN 2 имеет следующий формат:

PAINT <указатель координат>[,<цвет>]

Точка, определяемая указателем координат, должна находиться строго внутри раскрашиваемой области, но в то же время в пределах экрана. Кроме того, цвет раскраски должен соответствовать цвету границы области. Программа 8.13 рисует на экране круги, которые затем раскрашиваются в разные цвета.

Программа 8.13

```
10 REM *
20 REM * Закраска при высоком разрешении
30 REM *
40 COLOR 15,1,1
50 SCREEN 2
60 FOR C = 15 TO 1 STEP -1
70 CIRCLE (128,96),C*7,C
80 PAINT (128,96),C
90 NEXT
100 GOTO 100
```

Оператор PAINT работает так: начиная с некоторой внутренней точки, он покрывает поверхность определенным цветом до достижения границы того же цвета либо краев экрана. Если же цвета раскраски и границы области различны, происходит "расплескивание краски", т.е. цвет раскраски распространяется за пределы окрашиваемой поверхности. В программе 8.14 начальная точка, заданная оператором PAINT, оказывается вне области, что также вызывает "проливание" краски.

Программа 8.14

```
10 REM *
20 REM * Проливание краски
30 REM *
40 COLOR 15,1,1
50 SCREEN 2
60 LINE (10,10)-(20,20),8,BF
70 LINE (180,170)-(200,190),8,BF
80 CIRCLE (128,96),70,15
90 PAINT (10,10),15
100 GOTO 100
```

Если цвет пиксела, координаты которого фигурируют в операторе PAINT, совпадает с цветом раскраски, оператор "решает", что работа уже выполнена, и тогда, вопреки ожиданиям, область не будет раскрашена.

Раскрашивание при низком разрешении. В этом случае оператор PAINT имеет несколько иной вид (возможно включение дополнительного параметра):

PAINT <указатель координат>[,<цвет>][,<цвет границы>]

Под границей здесь подразумевается край раскрашиваемой области, а не экрана. Программа 8.15 изображает белую окружность и раскрашивает ее в белый цвет, а программа 8.16 такую же белую окружность закрашивает красным – получается красный круг с белой границей.

Программа 8.15

```
10 REM *
20 REM * Закраска при низком разрешении
30 REM *
40 COLOR 15,1,1
50 SCREEN 3
60 CIRCLE (128,96),70,15
70 PAINT (128,96),15,15
80 GOTO 80
```

Программа 8.16

```
10 REM *
20 REM * Закраска при низком
30 REM * разрешении с границей
40 REM *
50 COLOR 15,4,4
60 SCREEN 3
70 CIRCLE (128,96),70,15
80 PAINT (128,96),8,15
90 GOTO 90
```

При работе с этим оператором не стоит определять цвета по умолчанию. Цвет границы, фигурирующий в операторе PAINT, должен обязательно совпадать с цветом окрашиваемого контура. Попытка использовать оператор PAINT с параметром "цвет границы" в режиме SCREEN 2 приведет к синтаксической ошибке.

ОПРЕДЕЛЕНИЕ ЦВЕТА ПИКСЕЛА

Первая из рассматриваемых нами графических функций, POINT, возвращает код цвета указанной точки на графическом экране. В этой функции используются только абсолютные значения координат. Результатом является целое число от –1 до 15. Значение –1 возвращается в том случае, если заданная точка находится вне экрана.

POINT представляет собой одну из немногих связанных с графикой команд, обращение к которым даже в текстовом режиме не влечет за собой синтаксических ошибок, однако не стоит этим злоупотреблять. На текстовом экране POINT всегда будет возвращать нуль. Приведем пример применения функции POINT.

Программа 8.17 выводит на экран точки, координаты которых увеличиваются. Функция POINT позволяет определить, не выходят ли точки за пределы экрана, и при необходимости изменить направление и цвет вычерчиваемой линии. При достаточно длительной работе программы возникает изображение орнамента, напоминающего тканый узор, постоянно меняющийся по цвету.

Программа 8.17

```
10 REM *
20 REM * Тканые узоры
30 REM *
40 COLOR 1,1,1
50 SCREEN 2
60 COLOR 1,1,1
70 SCREEN 2
80 DX = 3 : DY = -3
90 X = 112 : Y = 96
100 C = 15 : R = 8
110 X = X + DX : Y = Y + DY
120 PSET (X,Y),C
130 IF POINT (X,Y) <> -1 GOTO 110
140 REM *
150 REM * Изменение цвета
160 REM * и поворот назад
170 REM *
180 DX = -DX
190 SWAP DX, DY
200 SWAP C, R
210 GOTO 110
```

СОЧЕТАНИЕ ТЕКСТА И ИЛЛЮСТРАЦИЙ

В большинстве программ для вывода информации используется оператор PRINT или какой-либо из его вариантов, например PRINT USING. При графическом экране применение PRINT не даст никакого эффекта. Столь же бесполезны в графическом режиме TRON и TROFF. Вместо этого следует открыть файл и записать его с помощью команды PRINT# . В качестве имени файла нужно указать символическое обозначение графического экрана "GRP:". После открытия файла можно записывать информацию на экран (но не считывать с него).

Место на экране, куда следует выводить данные, определяется оператором PRESET. Отмеченная им точка соответствует левому верхнему углу первого выводимого символа. Если позиция вывода не указана, данные будут выводиться начиная с последней задействованной позиции. Если текст должен располагаться за нижним правым углом экрана, "срабатывает" аналог команды HOME, и вывод продолжается с левого края верхней строки экрана.

Вывод текста на графический экран имеет одну особенность: задавая различные значения цвета основного изображения, вы можете менять цвета букв. Это иллюстрируется программой 8.18 и рис. 8.6.

В том случае, когда при выполнении программы необходимо обращение к другим файлам, например к кассетному магнитофону ("CAS:"), нужно позаботиться о своевременном увеличении параметра MAXFILES.

Программа 8.18

```
10 REM •
20 REM • Текст и изображения
30 REM •
40 COLOR 15,15,15
50 SCREEN 2
60 OPEN "GRP:" AS #1
70 LINE (0,0)-(255,191),1,B
80 LINE (120,0)-(120,191),1
90 PRESET (8,32)
100 FOR I = 1 TO 14
110 COLOR I
120 PRINT#1,"MSX COMPUTERS"
130 PRESET STEP (8,0)
140 NEXT I
150 PRESET (164,16)
160 COLOR 1
170 PRINT#1,"Круг"
180 CIRCLE (188,64),30
190 PAINT (188,64)
200 PRESET (144,112)
210 COLOR 8
220 PRINT#1,"Квадрат"
230 LINE (136,124)-STEP(40,40),8,BF
240 PRESET (186,112)
250 COLOR 4
260 PRINT#1,"Треугольник"
270 LINE (208,128)-STEP(30,30)
280 LINE -STEP(-30,0)
290 LINE -STEP(0,-30)
300 GOTO 300
```

ВВОД ДАННЫХ В ГРАФИЧЕСКОМ РЕЖИМЕ

Если при работе программы в графическом режиме встретится оператор INPUT, этот режим изменится на текстовый, поскольку INPUT генерирует знак "?". Следовательно, традиционный способ ввода данных в графическом режиме здесь непригоден. Возможные варианты решения проблемы связаны с применением функций или команд, обращающихся к клавиатуре: INKEY$, LINE INPUT или INPUT$. Выводя данные по мере их поступления на графический экран, пользователь может контролировать их. Программа 8.19 с помощью оператора INPUT$( ) выполняет ввод с дублированием эха на экране для режима с высокой степенью разрешения.

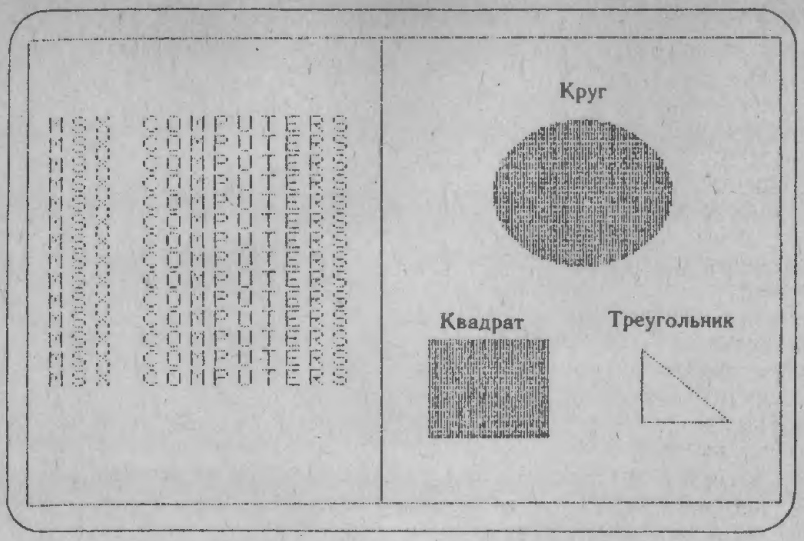

Рис. 8.6. Сочетание текста и иллюстраций (программа 8.18)

Программа 8.19

```
10 REM •
20 REM • Подпрограмма ввода
30 REM •
40 COLOR 15,15,15
50 SCREEN 2,0,0
60 OPEN "GRP:" AS #1
70 LINE (0,0)-STEP(255,191),1,B
80 PRESET (8,8)
90 COLOR 1
100 PRINT#1,"Радиус : ";
110 N$ = ""
120 X = 80
130 PRESET(X,8)
140 A$ = INPUT$(1)
150 IF A$ = CHR$(13) THEN GOTO 250
160 REM •
170 REM • Подтверждение ввода
180 REM •
190 IF A$<"0" OR A$>"9" THEN PLAY "L24O2C" : GOTO 130
200 PRINT#1,A$; : N$ = N$+A$ : X = X+8
210 GOTO 130
220 REM •
230 REM • Изображение окружности
240 REM •
```

```
250 R = VAL(N$)
260 IF R>80 THEN BEEP : GOTO 280
270 CIRCLE (128,100),R
280 LINE (80,8)-STEP(56,8),15,BF
290 GOTO 110
```

ОПЕРАТОР DRAW

Действие оператора DRAW, управляющего графикой, во многом аналогично действию оператора PLAY, регулирующего генерируемый программой звук. Оператор DRAW сопровождается строковым выражением, включающим набор букв – элементов еще одного подъязыка, так называемого графического макроязыка (ГМЯ, или GML). Как вы видели ранее, в частности, когда мы рассматривали относительные указатели координат, в MSX-Бейсике запоминается положение последней из задействованных точек, т.е. образуется своего рода невидимый графический курсор. При работе с оператором DRAW и его параметрами мы будем в значительной степени опираться на такое представление курсора.

Графические параметры. Основных параметров – восемь. После каждого из них указывается целое число, соответствующее количеству изображаемых элементов. Перечислим эти параметры:

U	вертикальная линия, направленная вверх;
R	горизонтальная линия, направленная вправо;
D	вертикальная линия, направленная вниз;
L	горизонтальная линия, направленная влево;
E	диагональная линия, направленная вверх и вправо;
F	диагональная линия, направленная вниз и вправо;
G	диагональная линия, направленная вниз и влево;
H	диагональная линия, направленная вверх и влево.

Главное достоинство команд ГМЯ состоит в том, что они позволяют быстро и точно описать сложные контуры. Кроме того, они предоставляют более естественный способ изображения линий, чем несколько абстрактные композиции из прямых и эллипсов. Эти команды показаны в программе 8.20, а результат их выполнения – на рис. 8.7.

Программа 8.20

```
10 REM *
20 REM * Команды ГМЯ
30 REM *
40 COLOR 15,1,1
50 SCREEN 2
60 REM *
70 REM * Лестница
80 REM *
90 PSET(20,42)
```

```
100 FOR I = 1 TO 4
110 DRAW "E40R20D15G40U15E40G40L20R20D15"
120 NEXT I
130 DRAW "L80U60"
140 PAINT STEP (1,1),15
150 DRAW "LIU1E40R80D60"
160 PAINT STEP (-1,-1),15
170 DRAW "R1DIG40"
180 COLOR 1
190 DRAW "U15"
200 COLOR 15
210 REM *
220 REM * Спираль
230 REM *
240 PSET(188,40)
250 DRAW "R5D5L10U10R15D15L20U20"
260 DRAW "R25D25L30U30R35D35L40"
270 REM *
280 REM * Домик
290 REM *
300 PSET (140,112)
310 DRAW "E15R30F15L60R5D40R5U30R15"
320 DRAW "D30L15R45U40"
330 PSET STEP (-25,10)
340 DRAW "R20D20L20U20D10R20L10U10D20"
350 PSET STEP (0,-45)
360 DRAW "L5U5R5D4"
370 GOTO 370
```

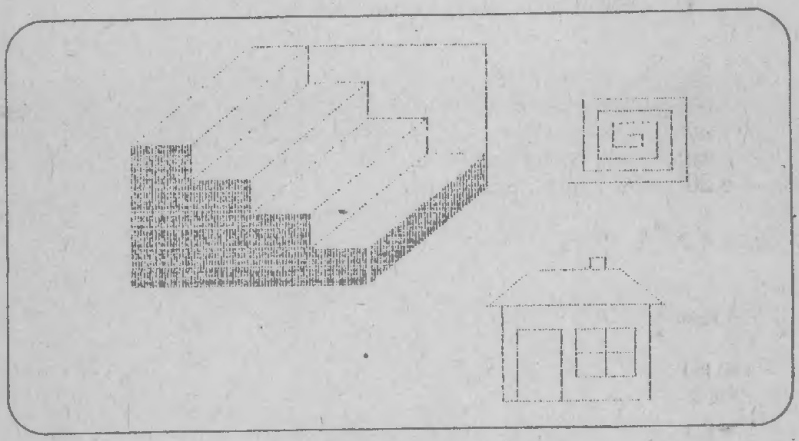

Рис. 8.7. Изображения, полученные по программе 8.20

Вместо команды LINE в ГМЯ используется параметр M. От текущего положения курсора проводится отрезок прямой линии до точки, координаты которой задаются в абсолютной или отрицательной форме. В последнем случае значениям по X и Y явно присваивается знак "+" или "–" даже тогда, когда одно из них равно нулю:

```
40 DRAW "M+100,+0"
```

Действие параметра M аналогично действию LINE в относительной форме.

Позиционирование курсора. Положение графического курсора можно задавать с помощью PSET и PRESET, но это несколько грубый способ. Обе команды могут изменить цвет пиксела, что в ряде случаев нежелательно. Если перед какой-либо из команд ГМЯ, выполняющих рисование или перемещение, поставить параметр B, то курсор переместится на нужное число точек, ничего не изменяя на экране. Таким способом можно указывать местоположение текста, выводимого на графический экран, что делает оператор DRAW особенно полезным.

Если же предпослать команде ГМЯ параметр N, движение по экрану будет сопровождаться рисованием следа, после чего графический курсор вернется в исходное положение. Действие параметров N и B иллюстрируется программой 8.21.

Программа 8.21

```
10 REM •
20 REM • Параметры N и B
30 REM •
40 OPEN "GRP:" AS #1
50 SCREEN 2
60 DRAW "BM16,8"
70 PRINT #1,"Возможности оператора DRAW"
80 DRAW "BM128,96"
90 DRAW "NU60NE40NR60NF40"
100 DRAW "ND60NG40NL60NH40"
110 REM •
120 REM • Восстановление положения курсора
130 REM •
140 DRAW "BM16,176"
150 PRINT #1,"ГМЯ"
160 GOTO 160
```

Установка цвета. Обычно в графических командах ГМЯ подразумевается текущее значение цвета. Однако любой из цветов, имеющихся в MSX-Бейсике, можно задать в строке ГМЯ с помощью параметра C, указав при нем аргумент в пределах от 0 до 15. Например, C15 устанавливает белый цвет изображения. В отличие от команды COLOR, установка цвета в ГМЯ не влияет на значения, используемые по умолчанию в таких графических командах, как PAINT или CIRCLE.

Растяжение и вращение. Параметр S позволяет растянуть или сжать любое полученное с помощью ГМЯ изображение. Рассмотрим, например, строку ГМЯ "R12". Казалось бы, эта запись должна означать следующее: "изобразить вправо 12 пикселов". Но надо учесть, что масштабирующий множитель в ГМЯ равен 1/4, т.е. перемещение на одну позицию трактуется как 1/4 пиксела. По умолчанию принимается S4 — единичное перемещение соответствует одному пикселу. Поэтому по команде "S2R12" будут изображены шесть пикселов вместо 12, а "S8R12" даст линию из 24 пикселов.

Если коэффициент масштабирования при сжатии или растяжении слишком велик, могут возникнуть затруднения. Рассмотрим такой оператор:

```
40 DRAW "S1R16D17R3U1"
```

Фактическим результатом его выполнения окажется изображение:

- вправо на 4 пиксела;
- вниз на 4 пиксела;
- вправо на 0 пикселов;
- вверх на 0 пикселов.

После округления "вниз" ряда параметров изображение будет существенно отличаться от ожидаемого. Эффект растяжения иллюстрируется программой 8.22.

Программа 8.22

```
10 REM •
20 REM • Растяжение изображения
30 REM •
40 SCREEN 2
50 COLOR 15,4,1
60 FOR I=4 TO 18
70 CLS
80 N=40+I : PLAY "N=N:"
90 DRAW "BM40,60"
100 DRAW "S"+STR$(I)
110 DRAW "R40D30L40U30BM+5,+30"
120 DRAW "U25R10D25BM+5,-20"
130 DRAW "R15D15L15U15BM+7,+0"
140 DRAW "D15BU7NR8L7BM+20,-18"
150 DRAW "R5M-10,-15L30M-10,15NR5"
160 DRAW "BM+40,+15"
170 FOR D=1 TO 250 : NEXT D
180 NEXT
190 GOTO 190
```

Установленный масштаб действует по умолчанию для всех последующих команд ГМЯ. В любой программе масштаб определяется первой командой DRAW.

Изложенное выше справедливо и для параметра A. Он задает угол поворота изображения против часовой стрелки. Это означает, что если, допустим, задано вращение на 90°, то параметр R будет действовать как U. Приведем список допустимых значений параметра A:

Параметр	Угол поворота против часовой стрелки
A0	0°
A1	90°
A2	180°
A3	270°

Используя параметры растяжения и вращения в различных сочетаниях, можно получить интересные геометрические узоры. На рис. 8.8 изображен узор, полученный при выполнении программы 8.23.

Программа 8.23

```
10 REM *
20 REM * Растяжение и повороты
30 REM *
40 COLOR 15,1,1
50 SCREEN 2
60 FOR S = 2 TO 8
70 FOR A = 0 TO 3
80 PSET (128,96)
90 DRAW "S=S:A=A:R30E15U30L30G15D30E15"
100 DRAW "R30L30U30D15L15R30E15G15"
110 DRAW "D30"
120 NEXT A
130 NEXT S
140 GOTO 140
```

Переменные и подстроки. Как и в музыкальном макроязыке, в ГМЯ имеется возможность включения переменных и подстрок. Соответствующие правила работы здесь те же, что и в ММЯ, т.е. после параметра ставится знак "=", имя переменной и точка с запятой.

С помощью параметра X часто встречающиеся команды ГМЯ можно оформить в виде подстрок. Еще один случай, когда подстроки необходимы, — применение очень длинных (более 255 знаков) команд. Используя X, такие команды можно разбивать на подстроки, длина которых уже не превышает допустимых размеров, и присваивать им имена.

ПРИМЕРЫ ПРОГРАММ

В этом разделе мы продемонстрируем вам действие графических команд MSX-Бейсика на нескольких конкретных примерах.

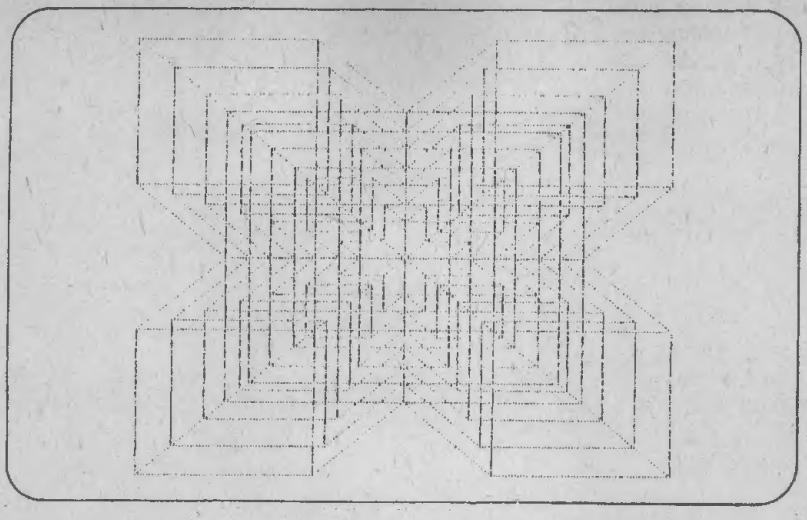

Рис. 8.8. Узор, полученный с помощью программы 8.23

Программа 8.24. Копирование изображений. Чтобы скопировать изображение из одной области экрана в другую, нужен двумерный массив. Первый этап работы состоит в определении размеров копируемой области и создании массива требуемой размерности.

Программа 8.24

```
10 REM •
20 REM • Копирование участка экрана
30 REM •
40 CLEAR
50 DEFINT A–Z
60 REM •
70 REM • Массив для графических данных
80 REM •
90 DIM A(50,50)
100 BC = 1 : SC = 15
110 COLOR 15,1,1
120 SCREEN 2
130 ON INTERVAL = 100 GOSUB 630
140 OPEN "GRP:" AS #1
150 LINE (56,0)–STEP(128,10),15,BF
160 PSET (68,1) : COLOR 1
170 PRINT#1,"Копирование экрана"
180 COLOR 15
190 REM •
200 REM • Изображение
```

```
210 REM *
220 LINE (80,80)-(130,130),15,BF
230 LINE (82,82)-(128,128),1,B
240 FOR I = 2 TO 12 STEP 2
250 CIRCLE (105,105),20,1,,,12/I
260 CIRCLE (105,105),20,1,,,I/12
270 NEXT I
280 REM *
290 REM * Заполнение массива
300 REM *
310 INTERVAL ON
320 FOR I = 1 TO 50
330 FOR J = 1 TO 50
340 A(I,J) = POINT (80+I,80+J)
350 NEXT J
360 NEXT I
370 INTERVAL OFF
380 COLOR ,,1
390 REM *
400 REM * Получение копий
410 REM *
420 FOR I = 1 TO 2
430 READ X,Y
440 FOR J = 1 TO 50
450 FOR K = 1 TO 50
460 PSET (X+J,Y+K),A(J,K)
470 NEXT K
480 NEXT J
490 NEXT I
500 REM *
510 REM * Перевернутое изображение
520 REM *
530 FOR I = 1 TO 50
540 FOR J = 1 TO 50
550 IF A(I,J)=1 THEN C=15 ELSE C=1
560 PSET (29+I,80+J),C
570 NEXT J
580 NEXT I
590 GOTO 590
600 REM *
610 REM * Подпрограмма интервалов
620 REM *
630 PLAY "T255L64SM1000O4CO5C"
640 SWAP BC,SC : COLOR ,,SC
650 RETURN
660 REM *
670 REM * Значения координат
680 REM *
690 DATA 30,28,80,28
```

В этом массиве хранятся значения цветов каждого пиксела копируемого изображения, которые находятся с помощью функции POINT. Затем эти данные используются в операторе PSET для переноса отмеченных точек в другое место экрана. Запоминание данных в массиве полезно с нескольких точек зрения. Изменяя систему индексирования, можно перевернуть изображение на 180 градусов или симметрично отразить его от осей координат. Если при работе используются лишь два цвета, их легко поменять местами.

Заметим, что в массиве чтение и запись данных выполняются невыносимо медленно, поэтому нужно запастись терпением.

Программа 8.25. Зеркальное отражение. Эта программа генерирует эффекты, подобные калейдоскопическим. С помощью клавиш управления курсором на экране отмечается след — набор точек, которые отражаются относительно осей X и Y, причем координаты берутся соответственно из диапазонов (0,96) –(255,96) и (128,0) – (128,191). Установка положения курсора производится с использованием двумерного массива и функции STICK( ). В массиве хранятся значения сдвигов, необходимые для перемещения курсора в направлении, определенном функцией STICK( ). Такой способ установки курсора значительно быстрее, чем с помощью последовательности из восьми условных операторов IF...THEN и даже чем ветвление ON GOTO.

Программа 8.25

```
10 REM *
20 REM * Зеркальные отражения
30 REM *
40 COLOR 15,1,1
50 SCREEN 2,0,0
60 DIM DIR(9,2)
70 X = 128 : Y = 96
80 REM *
90 REM * Направления джойстика
100 REM *
110 FOR I = 0 TO 8
120 FOR J = 0 TO 1
130 READ DIR(I,J)
140 NEXT J
150 NEXT I
160 REM *
170 REM * Основной цикл
180 REM *
190 X = X + DIR(STICK(0),0)
200 Y = Y + DIR(STICK(0),1)
210 REM *
220 REM * Проверка границ
230 REM *
240 IF X > 255 THEN X = 255
250 IF Y > 191 THEN Y = 191
```

```
260 IF X < 0 THEN X = 0
270 IF Y < 0 THEN Y = 0
280 REM *
290 REM * Изображение точек
300 REM *
310 PSET(X,Y)
320 PSET(256–X,Y)
330 PSET(X,192–Y)
340 PSET(256–X,192–Y)
350 GOTO 190
360 DATA 0,0,0,–1,1,–1,1,0,1,1
370 DATA 0,1,–1,1,–1,0,–1,–1
```

Программа 8.26. Движение секундной стрелки. В гл.3 рассматривалась программа поворота точки на определенный угол, а в гл.4 — прерывания типа ON INTERVAL. Ниже эта методика применяется для моделирования движения секундной стрелки. Используемый в программе массив содержит координаты последовательных положений точки оси Y при поворотах ее на углы от 0 до 360°. Таким образом, на циферблате изображается перемещающаяся секундная стрелка.

Программа 8.26

```
10 REM *
20 REM * Изображение окружности
30 REM * – "циферблата"
40 REM *
50 COLOR 15,1,1
60 DIM T(60,2)
70 ON INTERVAL=50 GOSUB 310
80 SCREEN 2
90 X = 0 : Y = 60
100 IX=0 : M$="T255L24S1M800O6C"
110 FOR I = 0 TO 6.17847 STEP .10471
120 REM *
130 REM * Описание массива
140 REM * и отметки секунд
150 REM *
160 NX =(X * COS(I))-(Y * SIN(I))
170 NY =(X * SIN(I))+(Y * COS(I))
180 A = (128 – (NX * .9))
190 B = 96 – NY
200 MA=A+SGN(A–128):MB=B–SGN(96–B)
210 CIRCLE(MA,MB),1
220 T(IX,0)=A : T(IX,1)=B
230 IX=IX+1
240 NEXT I
250 IX=59 : INTERVAL ON
260 GOTO 260
```

```
270 REM •
280 REM • Ежесекундные
290 REM • перемещения стрелки
300 REM •
310 IF IX<59 THEN 360
320 LINE(128,96)-(T(59,0),T(59,1)),1
330 IX=0 : FLAY M$
340 LINE(128,96)-(T(IX,0),T(IX,1)),15
350 RETURN
360 LINE(128,96)-(T(IX,0),T(IX,1)),1
370 IX=IX+1 : PLAY M$
380 LINE(128,96)-(T(IX,0),T(IX,1)),15
390 RETURN
```

Программа 8.27. Круговые диаграммы. Круговые диаграммы можно изображать методом поворота точки, однако быстрее и удобнее те же действия выполняет оператор CIRCLE с параметрами "начальный угол" и "конечный угол". Предусматривается изображение до 12 элементов, но при необходимости это число легко изменить. Характерный пример круговой диаграммы приведен на рис. 8.9.

Программа 8.27

```
10 REM •
20 REM • Круговые диаграммы
30 REM •
40 OPEN "GRP:" AS #1
50 COLOR 15,15,15
60 DIM A(12)
70 V=0
80 FOR I=1 TO 12
90 READ A(I)
100 V = V+A(I)
110 NEXT I
120 REM •
130 REM • Расчет параметров для
140 REM • построения диаграммы
150 REM •
160 SCREEN 2
170 UNIT = 6.283/V
180 LINE (0,0)-(255,191),1,B
190 LINE (4,4)-(80,187),1,BF
200 LINE (84,4)-(251,187),1,BF
210 LINE (86,6)-(249,185),15,B
220 CIRCLE (176,96),80,15,,,1.2
230 CIRCLE (176,96),80,15,-6.283,-6.283,1.2
240 SA = 0
250 FOR I=1 TO 12
260 SA=SA+(UNIT*A(I))
```

```
270 CIRCLE (176,96),80,15,–SA,–SA,1.2
280 NEXT
290 LINE (6,6)-(78,185),15,B
300 LINE (6,22)–STEP(72,0),15
310 DRAW "BM24,12":PRINT#1,"Данные":PRINT#1,STRING$(3,13)
320 FOR I=1 TO 12
330 PRINT#1,USING " ###.";I;
340 PRINT#1, USING "####";A(I)
350 NEXT
360 GOTO 360
370 DATA 100,80,66,90,89,14
380 DATA 150,120,72,35,16,94
```

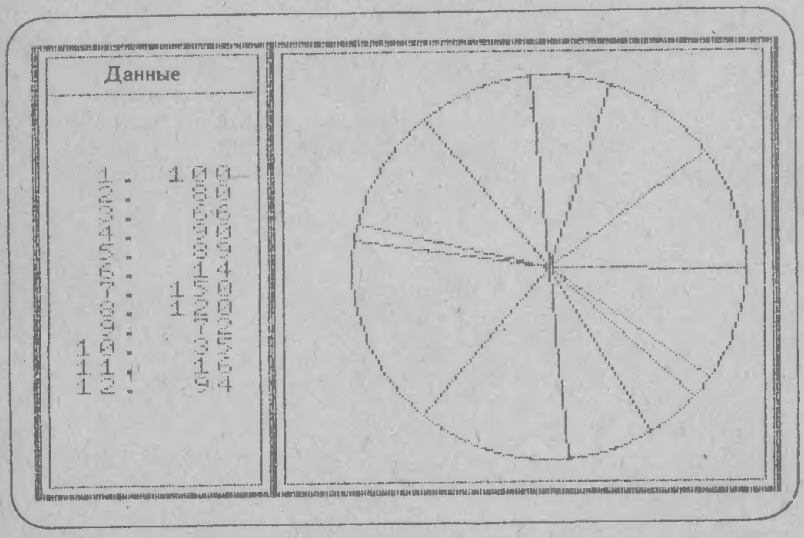

Рис.8.9. Круговая диаграмма, результат выполнения программы 8.27

Программа 8.28. Графики. Настоящий раздел завершается программой, позволяющей получать полный набор графиков: в виде прямоугольных гистограмм, точек и ломаных линий. Координатные оси размечаются так же, как на специальной бумаге для графиков, причем максимальное и минимальное значения данных определяют масштаб по оси Y и положение оси X.

Программа 8.28

```
10 REM ************************
20 REM *                      *
30 REM * Построение графиков   *
40 REM *                      *
50 REM ************************
```

```
60 REM *
70 REM * К подпрограмме ввода
80 REM *
90 GOTO 200
100 REM *
110 REM * Прокрутка вводимых данных
120 REM *
130 LOCATE 0,4,0
140 FOR I = ITEM-18 TO ITEM-1
150 PRINT USING "###";I;:PRINT TAB(7); TBL(I)
160 NEXT I
170 LOCATE 3,22 : PRINT SPC(16)
180 LOCATE 0,22
190 RETURN
200 REM *
210 REM * Раздел ввода данных
220 REM *
230 SCREEN 0
240 OPEN "GRP:" FOR OUTPUT AS #1
250 PLAY "T255L12SM2000"
260 COLOR 15,4,4 : KEY OFF
270 DIM TBL(110)
280 CLS
290 PRINT "Ввод данных:" : PRINT
300 PRINT "Максимальное из 110 значений
(* - выход)." : PRINT
310 ITEM = 1 : MX=0 : MIN=0
320 PRINT USING "###";ITEM;: PRINT ":"; TAB(8)
330 LINE INPUT A$
340 REM *
350 REM * Прекращение ввода данных по "*"
360 REM *
370 IF A$="*" THEN GOTO 570
380 X=VAL(A$)
390 REM *
400 REM * Заполнение массива и определение
410 REM * максимального и минимального элементов
420 REM *
430 TBL(ITEM)=X : IF X>MX THEN MX=X
440 IF X<MIN THEN MIN = X
450 ITEM = ITEM + 1
460 REM *
470 REM * Циклическая прокрутка,
480 REM * если экран заполнен
490 REM *
500 IF ITEM>19 THEN GOSUB 130
510 IF ITEM < 111 THEN GOTO 320
520 PRINT
530 PRINT "Массив заполнен.Для вызова меню нажмите клавишу."
```

```
: X$ = INPUT$(1)
540 REM *
550 REM * Главное меню
560 REM *
570 SCREEN 0
580 PRINT TAB(1):"Виды графиков"
590 PRINT : PRINT
600 PRINT TAB(8):"1. Гистограмма" : PRINT
610 PRINT TAB(8):"2. Линейный график" :PRINT
620 PRINT TAB(8):"3. Точечная диаграмма" :PRINT
630 PRINT TAB(8):"4. Ввод данных":PRINT
640 PRINT TAB(8):"5. Выход из программы":PRINT
650 PRINT TAB(8):"Выберите номер (1–5): ( )";
651 LOCATE 2,20,0:PRINT "* Для возврата в меню
нажмите пробел"
660 BL$ = " ": BC$ = "_"
670 LOCATE 29,13,0
680 A$=INKEY$
690 IF A$<>"" THEN 740
700 SWAP BL$,BC$
710 PRINT BL$;
720 FOR D = 1 TO 100 : NEXT D
730 GOTO 670
740 PRINT A$
750 REM *
760 REM * Контроль ввода
770 REM *
780 IF A$<"1" OR A$>"5" THEN PLAY "O2F"
: LOCATE 29,13: PRINT " ": GOTO 670
790 PLAY "O4CO6C"
800 PRINT : PRINT
810 PRINT TAB(8):"Выбран":A$:" вариант."
820 FOR D = 1 TO 500 : NEXT
830 SEL = VAL(A$)
840 IF SEL = 4 THEN GOTO 280
850 IF SEL = 5 THEN CLS : END
860 CLS
870 COLOR 15,4,4
880 SCREEN 2
890 REM *
900 REM * Вычисление масштабного множителя
910 REM * для построения графика
920 REM *
930 R = ABS(MX)+ABS(MIN)
940 IF R = 0 THEN GOTO 570
950 FTR = 176/R
960 IF NOT(SGN(MIN)) THEN OG = 183 : GOTO 1010
970 REM *
980 REM * Начало координат и оси
```

```
990 REM *
1000 OG =183 + (MIN*FTR)
1010 LINE (15,OG)-STEP(240,0)
1020 LINE (15,0)-STEP(0,191)
1030 CIRCLE(8,OG),2
1040 REM *
1050 REM * Масштаб по Y ^10
1060 REM *
1070 GOSUB 1560
1080 IF OG = 183 THEN GOTO 1130
1090 PSET (16,OG)
1100 FOR I = 0 TO INT(ABS(MIN)/10^P)
1110 LINE (12,OG+(FTR*(10^P)*I))-STEP(3,0)
1120 NEXT
1130 PSET (16,OG)
1140 FOR I = 0 TO INT(MX/10^P)
1150 LINE (12,OG-(FTR*(10^P)*I))-STEP(3,0)
1160 NEXT
1170 REM *
1180 REM * Определение шага
1190 REM * по оси X
1200 REM *
1210 STP =INT(224/(ITEM-1))
1220 REM *
1230 REM * Выбор и изображение графика
1240 REM *
1250 ON SEL GOTO 1290,1390,1490
1260 REM *
1270 REM * Изображение гистограммы
1280 REM *
1290 SP = 15 : PSET(SP,OG)
1300 FOR I = 1 TO ITEM-1
1310 LINE (SP,OG)-STEP(STP-2,
-(TBL(I) * FTR)),,BF
1320 SP =SP + STP
1330 NEXT I
1340 FOR D=1 TO 500:NEXT
1350 X$ = INPUT$(1) : GOTO 570
1360 REM *
1370 REM * Изображение линейного графика
1380 REM *
1390 SP = 15 : PSET(SP,OG)
1400 FOR I = 1 TO ITEM-1
1410 LINE -(SP,OG-(TBL(I)*FTR))
1420 SP = SP + STP
1430 NEXT I
1440 FOR D=1 TO 500:NEXT
1450 X$ = INPUT$(1) : GOTO 570
1460 REM *
```

```
1470 REM * Изображение точечной диаграммы
1480 REM *
1490 SP = 20 : PSET(SP,OG)
1500 FOR I = 1 TO ITEM-1
1510 CIRCLE(SP,OG-(TBL(I)*FTR)),1
1520 SP = SP + STP
1530 NEXT I
1540 X$ = INPUT$(1) : GOTO 570
1550 FOR D=1 TO 500:NEXT
1560 REM *
1570 REM * Определение масштаба
1580 REM * по оси Y ^ 10
1590 REM *
1600 P=4
1610 IF R <10 THEN P = 0 : RETURN
1620 IF INT(R/10^P) = 0 THEN P = P-1:
GOTO 1620
1630 RETURN
```

Интервалы по оси X выбираются постоянными, что исключает в описываемой реализации возможность построения графиков "научного типа", с параллельным расположением аргумента и функции. Типичные примеры графиков приведены на рис. 8.10 а,б,в.

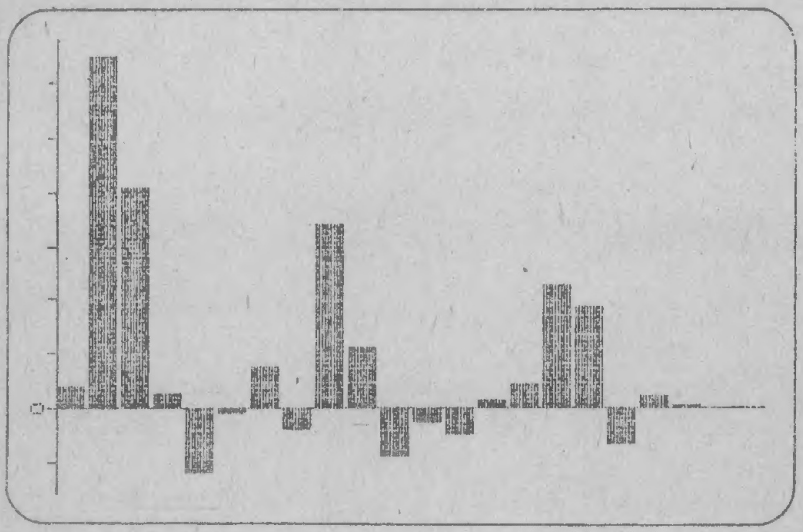

Рис. 8.10а. Программа 8.28 – гистограмма

Рис. 8.10б. Программа 8.28 – точечная диаграмма

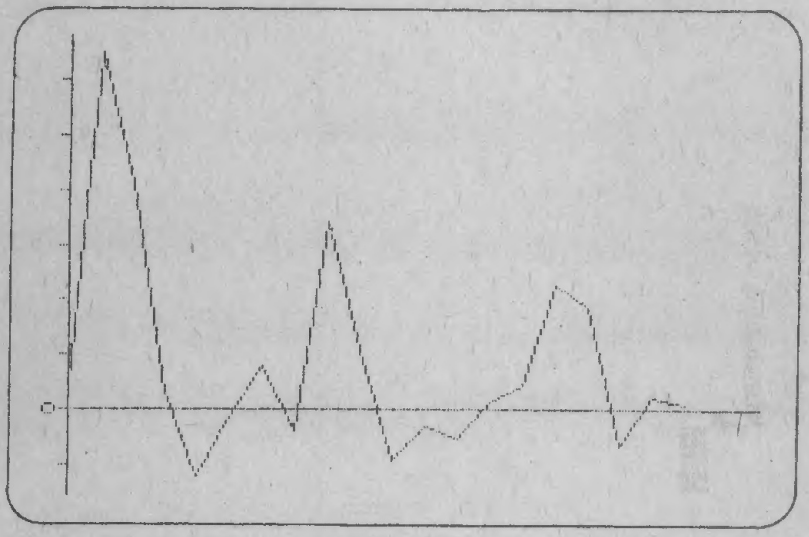

Рис. 8.10в. Программа 8.28 – линейный график

СВОДКА СЛУЖЕБНЫХ СЛОВ

Указатели координат

STEP(<сдвиг по X>,<сдвиг по Y>)
(<X абсолютное>,<Y абсолютное>)

Общие команды

PSET <указатель координат>[,<цвет>]
PRESET <указатель координат>[,<цвет>]
LINE [<указатель координат>]–<указатель координат>
[,<цвет>][B[BF]
CIRCLE <указатель координат>,<радиус>[,<цвет>][,<начальный угол>][,<конечный угол>][,<овал>]
PAINT <указатель координат>[,<цвет>][,<цвет границы>]
POINT(<X>,<Y>)

Параметры оператора DRAW

Общие графические команды

U,R,D,L,E,F,G,H

Команды изображения линий

M <X>,<Y>
M <сдвиг по X>,<сдвиг по Y>

Префиксы

B движение курсора без рисования следа
N движение курсора с рисованием следа и возвратом его в начальное положение

Прочие

C установка текущего значения цвета
S установка текущего значения масштаба
A установка текущего значения угла поворота
X выполнение графической подпрограммы

Глава 9. ДОПОЛНИТЕЛЬНЫЕ ГРАФИЧЕСКИЕ СРЕДСТВА

В настоящей главе описываются команды MSX-Бейсика, с помощью которых пользователь может по своему желанию создавать графические фигуры и образы и перемещать их (манипулировать ими). Кроме того, разбираются некоторые детали прямого доступа к видеопроцессору и поддерживающей его работу видеопамяти.

"ОЖИВЛЕНИЕ" ИЗОБРАЖЕНИЙ

Для создания иллюзии движения объекта достаточно выполнить следующие простые действия:

1. Изобразить объект в точке X, Y.
2. Уничтожить объект, окрасив его в цвет фона.
3. Увеличить или уменьшить значения X, Y.
4. Перейти к п.1.

Этот подход иллюстрируется программой 9.1, которая заставляет окружность кататься по экрану вперед и назад. Результат работы программы вряд ли можно считать идеальным: процессы создания и уничтожения изображения проходят не настолько гладко, чтобы дать полную иллюзию движения. Другой недостаток такого метода обнаруживается, когда движущийся объект попадает на участок экрана, не окрашенный в цвет фона. На втором этапе качество изображения будет ухудшаться. Преодолеть указанное препятствие можно лишь, сохранив в памяти параметры области, перекрываемой объектом, и восстановив их после его удаления.

Программа 9.1

```
10 REM *
20 REM * Простое "оживление"
30 REM *
40 COLOR 15,4,4
50 SCREEN 2
60 X=5:SX=5
70 CIRCLE (X,96),10,15
```

```
80 REM *
90 REM * Основной цикл
100 REM *
110 CIRCLE (X,96),10,4
120 X=X+SX
130 IF X<5 OR X>250 THEN SX=-SX
140 CIRCLE (X,96),10,15
150 GOTO 110
```

Рассмотренная процедура выполняется недопустимо медленно. К счастью, видеопроцессор располагает встроенным средством, которое позволяет "оживлять" изображение путем применения спрайтов – объектов, определяемых пользователем.

СВОЙСТВА СПРАЙТОВ

Перечислим характеристики спрайтов:

1. Равномерность движения. Спрайты могут перемещаться по экрану без эффектов дрожания и мерцания и не нарушают фон.

2. Трехмерность. На экране создается впечатление перспективы: спрайты появляются друг перед другом или один позади другого.

3. Возможность контроля столкновений. Видеоплата может фиксировать столкновение на экране двух и более спрайтов. Это открывает широчайшие возможности программирования разнообразных игр. В частности, при соприкосновении спрайтов, изображающих снаряд и космический корабль, можно всегда имитировать взрыв.

4. Автоматическая цикличность движения. Спрайт, покидающий пределы экрана, появляется на его противоположной стороне.

В системе MSX изображение можно "оживлять" на трех типах экранов: на текстовых экранах 32 * 24 и на графических экранах с высокой и низкой разрешающей способностью. На экране одновременно могут находиться не более 32 спрайтов одного из двух возможных размеров.

Создание спрайтов 8 * 8. По умолчанию спрайт в MSX-Бейсике имеет размеры 8 * 8 пикселов. В библиотеке можно хранить до 256 различных спрайтов такого размера. Рассмотрим последовательность действий при кодировании модели спрайта. Прежде всего требуемую форму (в данном случае крестообразный курсор) нужно изобразить на клетчатой бумаге 8 * 8, как показано на рис. 9.1. Перемещаясь сверху вниз по строкам такой решетки и обозначая пустые элементы нулями, а закрашенные – единицами, мы сводим макет рисунка к набору двоичных чисел:

```
00111000
00010000
10010010
11101110
10010010
00010000
00111000
00000000
```

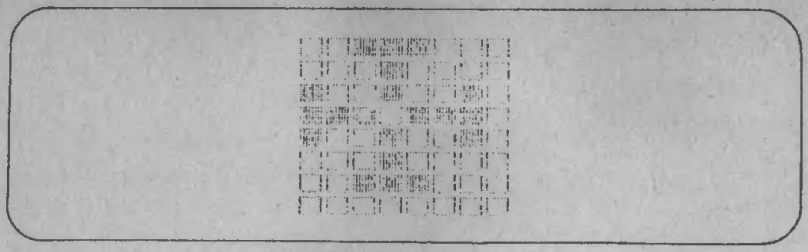

Рис. 9.1. Создание крестообразного курсора

Эти числа составляют основу описания модели спрайта. Их можно оставить в двоичном представлении, а если объем памяти ограничен — привести к двоичному или шестнадцатиричному виду.

Следующий шаг заключается в выборе типа экрана. Для удобства еще раз приведем формат оператора SCREEN:

SCREEN <режим>[,<размер спрайта>][,<звук клавиши>]
[,<скорость ленты>][,<параметры печати>]

Здесь необходимо правильно выбрать значения параметров "режим" и "размер спрайта". Можно устанавливать любой тип экрана, кроме текстового 40 * 24, а размеры спрайта могут быть следующими:

0	8 * 8	обычный (по умолчанию)
1	8 * 8	увеличенный
2	16 * 16	обычный
3	16 * 16	увеличенный

Увеличение в рассматриваемом случае означает удвоение видимых размеров спрайта. В нашем примере следует в качестве размера спрайта выбрать "0" или "1".

Фактический шаблон спрайта устанавливается, когда заданные величины считываются в особую переменную SPRITE$. Переменную SPRITE$ можно рассматривать как массив, каждый элемент которого содержит строку шаблона спрайта. Поскольку SPRITE$ имеет строковый тип, числовые данные перед присвоением необходимо преобразовать с помощью функции CHR$( ). Каждому шаблону приписывается индекс, или, как мы будем говорить, ***номер шаблона спрайта***. Чтобы описать нулевой шаблон, соответствующие данные нужно считать в SPRITE$(0). Это можно сделать, например, таким образом:

```
SPRITE$(0)  =  CHR$(56)+CHR$(16)+CHR$(146)+CHR$(238)+
               CHR$(146)+CHR$(16)+CHR$(56)+CHR$(0)
```

(Числа в скобках являются десятичными эквивалентами двоичных данных, описывающих рис. 9.1.) Аналогичное описание можно выполнить иначе — в цикле FOR...NEXT с использованием операторов READ и DATA.

Размещение спрайтов. Спрайт создается на экране по команде PUT SPRITE, формат которой приведен ниже:

PUT SPRITE <номер слоя>,<указатель координат>[,<цвет>]
[,<номер шаблона>]

Указатель координат определяет положение верхнего левого угла спрайта и может быть выражен в абсолютной или относительной форме, как и в других графических операторах.

В системе MSX все типы экранов, допускающих работу со спрайтами, можно рассматривать как множество *слоев* или *плоскостей*. Параметр "номер слоя" задает плоскость или слой, куда помещается спрайт. В одном слое может находиться лишь один спрайт (рис. 9.2). Спрайт нулевого слоя изображается поверх остальных спрайтов. Чем меньше номер слоя, тем выше его приоритет. В "модельном слое" спрайты располагать нельзя, там действуют графические команды, например LINE, CIRCLE и PAINT. Легко понять, почему при такой структуре слоев спрайты не оказывают никакого влияния на данные, имеющиеся на экране.

Если в операторе PUT SPRITE опущен параметр "номер шаблона", в MSX-Бейсике предполагается, что он совпадает с номером слоя.

В программе 9.2 описывается спрайт, изображающий крестообразный курсор, после чего он помещается на экран.

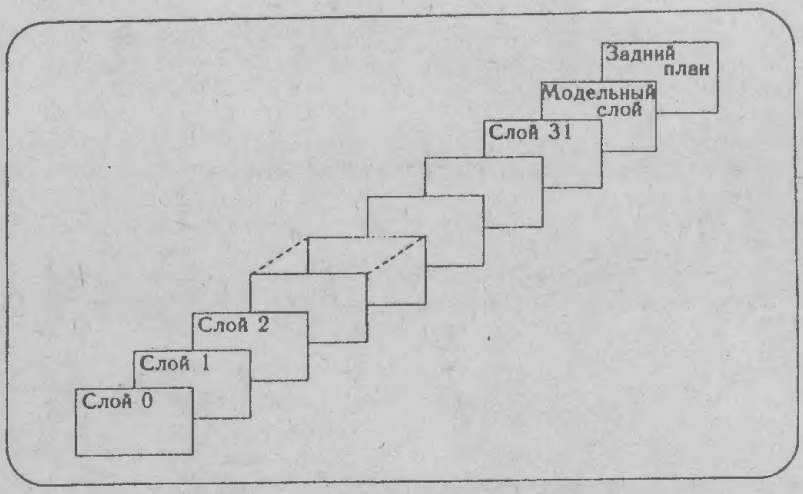

Рис. 9.2. Изображение слоев для спрайтов

Программа 9.2

```
10 REM *
20 REM * Движение спрайта
30 REM *
40 COLOR 15,4,4
50 SCREEN 2,0
60 FOR I=1 TO 8
70 READ A : S$=S$+CHR$(A)
80 NEXT
90 SPRITE$(0)=S$
100 INC = 1 : S = 20 : E = 235
110 FOR I = S TO E STEP INC
120 PUT SPRITE 0,(I,92),15,0
130 NEXT I
140 SWAP S,E : INC = -INC
150 GOTO 110
160 DATA &B00111000
170 DATA &B00010000
180 DATA &B00010000
190 DATA &B11101110
200 DATA &B10010010
210 DATA &B00010000
220 DATA &B00111000
230 DATA &B00000000
```

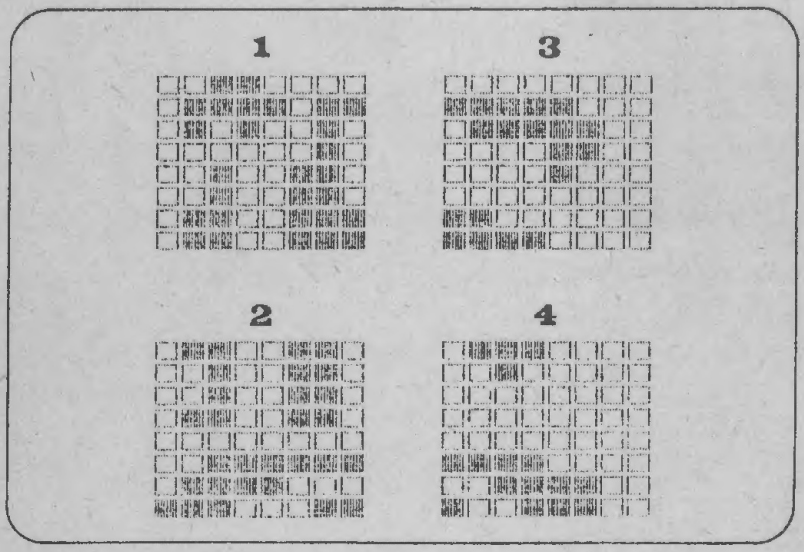

Рис. 9.3. Описание "увеличенного" спрайта

Если одновременно используется несколько спрайтов, для указания их положения не следует применять относительные координаты. В этом случае приращение координат берется относительно последнего из задействованных спрайтов, что делает весьма трудным, а то и вовсе невозможным независимое перемещение двух спрайтов.

Описание спрайтов 16 * 16. Чтобы описать спрайт размером 16 * 16 пикселов, необходимы все 32 элемента данных. Вновь на начальной стадии работы шаблон изображается в клетках, но числовые данные в этом случае определить несколько сложнее, чем для спрайтов 8 * 8. Решетка 16 * 16 разбивается на четыре блока 8 * 8, причем данные из них считываются в переменную SPRITE$ в следующем порядке: верхний левый блок, затем нижний левый, верхний правый и, наконец, нижний правый.

Спрайт, изображенный на рис. 9.3, описывается в программе 9.3. Отметим, что в операторе SCREEN размер спрайта выбирается равным 2. Как видите, для описания больших спрайтов требуется гораздо больше данных, вследствие чего число таких спрайтов ограничено 64. Как и их "меньшие собратья", они также могут быть удвоенного размера.

Программе 9.3

```
10 REM *
20 REM * Спрайт 16 * 16
30 REM * Буква "E" – готическое
40 REM *
50 COLOR 1,15,15
60 SCREEN 2,2
70 FOR I=1 TO 16
80 READ A$:S$=S$+CHR$(VAL("&B"+LEFT$(A$,8)))
90 NEXT
100 RESTORE 180
110 FOR I=1 TO 16
120 READ A$:S$=S$+CHR$(VAL"&B"+RIGHT$(A$,8)))
130 NEXT
140 SPRITE$(0)=S$
150 LINE (124,92)-(183,131),15,B
160 PUT SPRITE 0,(112,80),1,0
170 GOTO 170
180 DATA 0011000000000000
190 DATA 0111011111111000
200 DATA 0101001001111100
210 DATA 0000001000001100
220 DATA 0010011000001000
230 DATA 0010011000000000
240 DATA 0110011111000000
250 DATA 0110011111110000
260 DATA 0110011001110000
270 DATA 0010011000100000
280 DATA 0010011000000000
290 DATA 0110011000000000
300 DATA 0000000000000000
```

```
310 DATA 00111111110000
320 DATA 01110000011100
330 DATA 11100011001100
```

Правило пятого спрайта. При работе со спрайтами это наиболее существенное ограничение. Правило гласит:

На каждой горизонтальной строке могут одновременно находиться лишь четыре спрайта.

Если в строку помещается пять или больше спрайтов, видны будут только четыре из них с более высоким приоритетом (т.е. с наименьшими номерами слоев). Правило пятого спрайта наглядно демонстрирует программа 9.4.

Программа 9.4

```
10 REM *
20 REM * Правило пятого спрайта
30 REM *
40 COLOR 15,4,4
50 SCREEN 2,1
60 SPRITE$(0)=STRING$(8,255)
70 PUT SPRITE 1,(48,96),8,0
80 PUT SPRITE 2,(80,96),9,0
90 PUT SPRITE 4,(128,96),10,0
100 PUT SPRITE 5,(156,96),11,0
110 Y=16:SY=.5
120 PUT SPRITE 3,(104,Y),1,0
130 Y=Y+SY
140 IF Y<16 OR Y>176 THEN SY=-SY : PLAY
"L24O5CO6C"
150 GOTO 120
```

Вывод спрайтов за границы экрана и их уничтожение. Координаты, определяющие положение спрайтов, должны быть целочисленными, но из всех возможных значений смысл имеют лишь числа от –32 до 255 для горизонтальной координаты и от –32 до 191, а также 208 и 209 – для вертикальной координаты. Отрицательные значения указывают на то, что левая или верхняя часть спрайта выходит за границы экрана. Аналогично при значениях X и Y соответственно 255 и 191 спрайт полностью исчезает за правой или нижней границей экрана. В том случае, когда значение Y равно 208, все спрайты меньшего приоритета делаются невидимыми. Их можно восстановить на экране, если присвоить Y значение, отличное от 208. При Y, равном 209, спрайт полностью удаляется с экрана.

Управление столкновениями спрайтов. Два спрайта считаются столкнувшимися, если перекрываются хотя бы по одному их пикселу. При этом в Бейсике возникает прерывание, обрабатываемое оператором ON SPRITE GOSUB. Обработка происходит так же, как было рассмотрено в гл. 4. Режим ловушки включается по команде SPRITE ON. Если

при выполнении подпрограммы встречается оператор SPRITE STOP, обработка прерываний приостанавливается, однако после выхода из подпрограммы вновь устанавливается SPRITE ON. Обычно программист включает операторы SPRITE OFF и SPRITE ON в свои собственные программы обработки прерываний.

При прерываниях такого типа нет возможности выяснить, какие именно спрайты столкнулись, в то время как, допустим, ON KEY GOSUB позволяет понять, какая функциональная клавиша была нажата. Поэтому программисту следует так структурировать программу, чтобы быть в состоянии идентифицировать каждый случай столкновения (см. программу 9.5).

Программа 9.5

```
10 REM ******************
20 REM *                 *
30 REM * Игра в спрайты  *
40 REM * с прерываниями  *
50 REM *                 *
60 REM ******************
70 DEFINT A-Z
80 OPEN "GRP:" AS #1
90 DIM M(100)
100 DIM T(7)
110 R=RND(-TIME)
120 ON STRIG GOSUB 1000
130 ON SPRITE GOSUB 1250
140 ON INTERVAL=10 GOSUB 1110
150 ON SPRITE GOSUB 1250
160 COLOR 15,1,1
170 SCREEN 2,0,0
180 REM *
190 REM * Начальная установка спрайтов
200 REM *
210 FOR I = 0 TO 3
220 S$ = ""
230 FOR J = 1 TO 8
240 READ S : S$=S$+CHR$(S)
250 NEXT J
260 SPRITE$(I)=S$
270 NEXT I
280 REM *
290 REM * Изображение фона
300 REM *
310 DRAW "BM0,176"
320 DRAW "S4A0"
330 LINE(0,0)-(255,191),15,B
340 LINE(16,2)-(239,2),15
350 LINE(16,2)-(239,189),15,B
```

```
360 LINE(16,16)-(239,16),15
370 DRAW "BM136,5":COLOR 15
380 T$="Счет: ":GOSUB 920
390 SC = 0
400 PRINT#1,USING "######";SC
410 REM •
420 REM • Изображение звезд
430 REM •
440 FOR I=1 TO 100
450 E = INT(RND(1)•224)+16
460 F = INT(RND(1)•160)+16
470 PSET(E,F)
480 NEXT I
490 REM •
500 REM • Здесь случайным образом задается
510 REM • движение корабля противника
520 REM •
530 FOR I=1 TO 100
540 M(I)=INT(RND(1)•130)+16
550 NEXT I
560 CIRCLE(160,60),40,11:PAINT (160,60),11
570 CIRCLE(175,45),10,9,,,13/10:PAINT (175,45),9
580 REM •
590 REM • Установка отметок
600 REM •
610 DRAW "BM10,4"
620 FOR I=10 TO 13
630 PUT SPRITE I,STEP(10,0),15,1
640 NEXT
650 X=24:Y=96:LI=14
660 F$="T180V12O2S11M1000E16R16D#2"
670 M$="T180V12O6L24BGBGAB"
680 W$="T180S1M3000O3C"
690 MX=200:MY=96:IX=IX+1:T=-2:F=0:J=0
700 STRIG(0) ON
710 SPRITE ON
720 INTERVAL ON
730 IF IX>100 THEN IX=0
740 L = SGN(M(IX)-MY)
750 REM •
760 REM • Основной цикл
770 REM •
780 PUT SPRITE 0,(X,Y),15,1
790 PUT SPRITE 1,(MX,MY),15,0
800 IF F THEN GOSUB 1040
810 IF J THEN GOSUB 1160
820 IF STICK(0)=1 THEN Y=Y-2
830 IF STICK(0)=5 THEN Y=Y+2
840 IF Y<=16 THEN Y=16
```

```
850 IF Y>=176 THEN Y=176
860 MY=MY+L:IF MY=M(IX) THEN IX=IX+1: GOTO 730
870 MX=MX-T:IF MX<40 OR MX>228 THEN T=-T
880 GOTO 780
890 REM *
900 REM * Подпрограмма вывода на экран
910 REM *
920 FOR I = 1 TO LEN(T$)
930 PRINT#1,MID$(T$,I,1):
940 DRAW "BM-2,0"
950 NEXT
960 RETURN
970 REM *
980 REM * Свой снаряд
990 REM *
1000 IF F THEN RETURN
1010 PLAY "L8S8M300O6C","L8S8M300O7C"
1020 A=Y:B=X+16:F=1
1030 RETURN
1040 B=B+7
1050 IF B>232 THEN PUT SPRITE 3,(0,209): F=0:RETURN
1060 PUT SPRITE 3,(B,A),15,2:RETURN
1070 RETURN
1080 REM *
1090 REM * Снаряд противника
1100 REM *
1110 IF J THEN RETURN
1120 PLAY "L16S14M800O2A"
1130 BX=MX-8:BY=MY+4
1140 PUT SPRITE 4,(BX,BY),8,3
1150 J=-1 : STRIG(0) ON:RETURN
1160 BX=BX-5
1170 IF BX<16 THEN BY=209: J=0
1180 PUT SPRITE 4,(BX,BY),8,3
1190 RETURN
1200 REM *
1210 REM * Подпрограмма обработки прерываний
1220 REM * Определение спрайтов
1230 REM * по отношению к их расположению
1240 REM *
1250 SPRITE OFF : INTERVAL OFF
1260 IF A+2>=MY AND A+5<=MY+7 AND B+8>=
MX THEN SI=30:PLAY M$:GOTO 1300
1270 IF BY+2>Y AND BY-7<Y AND BX<34 THEN
LI=LI-1:PLAY F$:GOTO 1300
1280 IF A+2>=BY AND A+5<=BY+7 AND BY+8>=
36 THEN SI=10:PLAY W$:GOTO 1400
1290 SPRITE ON:INTERVAL ON:STRIG(0) ON:
RETURN
```

```
1300 REM *
1310 REM * Вспышка на экране
1320 REM *
1330 FOR I=0 TO 50
1340 COLOR ,,RND(1)*16
1350 NEXT
1360 MY=96:MX=220
1370 REM *
1380 REM * Удаление снаряда
1390 REM *
1400 PUT SPRITE 3,(0,209):F=0:A=0:B=209
1410 PUT SPRITE 4,(0,209):J=0:BX=0:BY=209
1420 IX=IX+1
1430 REM *
1440 REM * Увеличение счета
1450 REM *
1460 DRAW "BM178,5":COLOR 1
1470 PRINT#1,USING "#######":SC
1480 SC=SC+SI
1490 DRAW "BM178,5":COLOR 15
1500 PRINT#1,USING "#######":SC
1510 COLOR ,,1
1520 IF LI=9 THEN GOTO 1590
1530 PUT SPRITE LI,(0,209)
1540 SPRITE ON:INTERVAL ON
1550 RETURN 730
1560 REM *
1570 REM * Конец игры
1580 REM *
1590 STRIG(0) OFF:INTERVAL OFF:SPRITE OFF
1600 PUT SPRITE 0,(0,209)
1610 PUT SPRITE 1,(0,209)
1620 PUT SPRITE 3,(0,209)
1630 PUT SPRITE 4,(0,209)
1640 LINE (24,60)-(232,76),15,BF
1650 DRAW "BM26,64":COLOR 1
1660 T$="ИГРА ОКОНЧЕНА. Хотите играть еще - нажмите UP"
1670 GOSUB 920
1680 FOR I=1 TO 5000:NEXT
1690 S=STICK(0) :IF S=0 THEN 1690
1700 IF S<>1 THEN 1720
1710 CLS : GOTO 310
1720 COLOR 15,4,4
1730 END
1740 REM *
1750 REM * Данные для спрайтов
1760 REM *
1770 DATA 0,60,66,255,66,255,66,60
1780 DATA 0,224,68,255,255,68,224,0
```

```
1790 DATA 0,64,34,63,34,64,0,0
1800 DATA 0,2,68,252,68,2,0,0
```

Предположим, что вы управляете космическим кораблем, который может перемещаться вверх и вниз. Вы можете обстреливать появившийся вражеский корабль, который также стреляет по вашему кораблю. Программа структурирована таким образом, что возможны лишь следующие столкновения спрайтов:

- вражеский корабль с вашим снарядом;
- ваш корабль с вражеским снарядом;
- вражеский снаряд с вашим снарядом.

При попадании в цель вражеского снаряда ваш корабль гибнет, и игра прекращается; если же вам удается одним из своих снарядов поразить противника — ваш счет увеличивается.

Игры такого типа безжалостно обнажают серьезную слабость Бейсика: это весьма "медленный" язык. Спрайты позволяют несколько увеличить скорость выполнения программы, но ясно, что без машинных кодов обойтись не удастся.

ПРОГРАММИРУЕМЫЙ ЭКРАН

Спрайты широко используются в программе 9.6. Создание графического экрана — занятие утомительное, и настоящая программа может помочь в этом деле. После окончания формирования изображения текст Бейсик-программы можно вызвать на экран либо записать на ленту (по команде SAVE). При необходимости программу с ленты можно вновь загрузить (LOAD) или слить с другой программой, хранящейся в памяти (MERGE).

Программа 9.6

```
10 GOTO 180
20 REM *
30 REM * Изображение дополнительных координат
40 REM * (Подпрограмма)
50 REM *
60 PUT SPRITE 0,(16,P),1,H
70 PUT SPRITE 1,(22,P),1,T
80 PUT SPRITE 2,(28,P),1,U
90 PUT SPRITE 3,(16,P+8),1,H
100 PUT SPRITE 4,(22,P+8),1,T
110 PUT SPRITE 5,(28,P+8),1,U
120 RETURN
130 REM **********************
140 REM *                    *
150 REM * Экранный генератор *
160 REM *                    *
170 REM **********************
180 REM *
```

```
190 REM • Генератор графических программ
200 REM •
210 SCREEN 0 : KEY OFF : WIDTH 38
220 DEFINT A–Z
230 MAXFILES = 2
240 DIM BUF(100,5)
250 REM •
260 REM • Открытие файла
270 REM •
280 PRINT "ЭКРАННЫЙ ГЕНЕРАТОР"
290 PRINT
300 PRINT "Убедитесь, что установлены PLAY и RECORD"
310 PRINT : PRINT
320 PRINT "Введите имя файла: ";
330 A$=INPUT$(1)
340 IF A$=CHR$(13) AND F$<>"" THEN 390
350 A$ =CHR$(8) AND POS(0)–1>16 THEN
LOCATE POS(0)–1:PRINT CHR$(32); :LOCATE
POS(0)–1:F$=LEFT$(F$,LEN(F$)–1):GOTO330
360 IF LEN(F$)=6 THEN GOTO 330
370 IF (A$>="A" AND A$<="Z") OR (A$>="a"
AND A$<="z") THEN F$=F$+A$ ELSE GOTO330
380 PRINT A$; : GOTO 330
390 PRINT : PRINT
400 PRINT USING "Открывается файл..";F$
410 PRINT : PRINT
420 OPEN F$ FOR OUTPUT AS #2
430 PRINT USING "Файл `&` Открыт";F$
440 PRINT : PRINT
450 PRINT "Для продолжения нажмите любую клавишу"
460 A$=INPUT$(1)
470 SCREEN 2,0,0
480 GOSUB 800
490 IX=–1:P=8:Q=168:X=128:y=96
500 H=1:T=2:U=8:HI=0:TI=9:UI=6
510 MX=1:MY=1:HX=8:HY=8:LX=0:LY=0
520 ON SPRITE GOSUB 1780 : SPRITE ON
530 PUT SPRITE 6,(X,Y),1,10
540 GOSUB 60
550 A$=INKEY$:IF A$="" THEN 550
560 IF A$=CHR$(127) THEN IX=–1:CLS
570 IF A$=CHR$(27) THEN GOTO 1320
580 IF A$="P" THEN GOSUB 910
590 IF A$="L" THEN GOSUB 1000
600 IF A$="U" THEN GOSUB 1160
610 D=STICK(0) : IF D=0 THEN 550
620 IF D=1 THEN Y=Y–MY
630 IF D=2 THEN Y=Y–MY:X=X+MX
640 IF D=3 THEN X=X+MX
```

```
650 IF D=4 THEN X=X+MX:Y=Y+MY
660 IF D=5 THEN Y=Y+MY
670 IF D=6 THEN Y=Y+MY:X=X-MX
680 IF D=7 THEN X=X-MX
690 IF D=8 THEN X=X-MX:Y=Y-MY
700 IF X>255 THEN X=255
710 IF Y>191 THEN Y=191
720 IF X<0 THEN X=0
730 IF Y<0 THEN Y=0
740 H=INT(X/100):T=(XMOD100)/10:U=X MOD 10
750 H1=INT(Y/100):T1=(YMOD100)/10:U1=Y MOD 10
760 GOTO 530
770 REM *
780 REM * Описание спрайтов
790 REM *
800 FOR I=0 TO 11
810 S$ = ""
820 FOR J=1 TO 8
830 READ A$ : S$=S$+CHR$(VAL("&H"+A$))
840 NEXT J
850 SPRITE$(I)=S$
860 NEXT I
870 RETURN
880 REM *
890 REM * Подпрограмма закраски
900 REM *
910 IF POINT(X,Y)=15 THEN RETURN
920 IF IX+1 = 500 THEN RETURN
930 IX=IX+1
940 PAINT(X,Y)
950 BUF(IX,0)=1:BUF(IX,1)=X:BUF(IX,2)=Y
960 RETURN
970 REM *
980 REM * Изображение прямой
990 REM *
1000 IF L=0 THEN 1070
1010 L=0
1020 LINE(X,Y)-(BUF(IX,1),BUF(IX,2))
1030 IF (X=BUF(IX,1)) AND (Y=BUF(IX,2))
THEN BUF(IX,0)=2:PUT SPRITE 7,(0,209):
SPRITE ON:RETURN
1040 BUF(IX,3)=X : BUF(IX,4)=Y
1050 PUT SPRITE 7,(0,209)
1060 BUF(IX,0)=3:SPRITE ON:RETURN
1070 IF IX+1=100 THEN RETURN
1080 IX=IX+1
1090 L=1:SPRITE OFF
1100 BUF(IX,1)=X:BUF(IX,2)=Y
1110 PUT SPRITE 7,(X-1,Y-2),1,11
```

```
1120 RETURN
1130 REM •
1140 REM • Отмена предыдущего действия
1150 REM •
1160 IX=IX-1
1170 CLS
1180 IF IX<=-1 THEN IX=-1:BUF(0,0)=0:
RETURN
1190 FOR I=0 TO IX
1200 ON BUF(I,0) GOSUB 1230,1250,1270
1210 NEXT
1220 RETURN
1230 PAINT(BUF(I,1),BUF(I,2))
1240 RETURN
1250 PSET(BUF(I,1),BUF(I,2))
1260 RETURN
1270 LINE(BUF(I,1),BUF(I,2))-(BUF(I,3),BUF(I,4))
1280 RETURN
1290 REM •
1300 REM • Генерация программы
1310 REM •
1320 SCREEN 0
1330 PRINT "Вывод текста на принтер (P)
1340 PRINT "Вывод текста на экран (S)
1350 PRINT:PRINT
1360 PRINT "Select ( ):
1370 LOCATE 8,4
1380 A$=INKEY$ : IF A$="" THEN 1380
1390 PRINT A$;
1400 IF A$<>"P" AND A$<>"S" THEN BEEP :
GOTO 1370
1410 IF A$="P" THEN OPEN "LPT:" AS #1
1420 IF A$="S" THEN OPEN "CRT:" AS #1
1430 CLS
1440 REM •
1450 REM • Запись программы
1460 REM •
1470 FOR D = 1 TO 2
1480 PRINT #D,"10 REM •"
1490 PRINT #D,USING"20 REM • Имя программы: &";F$
1500 PRINT #D,"30 REM •"
1510 PRINT #D,"40 SCREEN 2"
1520 LN = 50
1530 FOR I = 0 TO IX
1540 PRINT #D,MID$(STR$(LN),2);SPC(1):
1550 ON BUF(I,0) GOSUB 1610,1650,1690
1560 LN=LN+10
1570 NEXT I
1580 PRINT #D,MID$(STR$(LN),2);SPC(1): "
```

```
GOTO ":MID$(STR$(LN),2)
1590 NEXT D
1600 END
1610 T$=MID$(STR$(BUF(I,1)),2)
1620 U$=MID$(STR$(BUF(I,2)),2)
1630 PRINT #D, USING "PAINT(&,&)":T$;U$
1640 RETURN
1650 T$=MID$(STR$(BUF(I,1)),2)
1660 U$=MID$(STR$(BUF(I,2)),2)
1670 PRINT #D, USING "PSET(&,&)":T$;U$
1680 RETURN
1690 T$=MID$(STR$(BUF(I,1)),2)
1700 U$=MID$(STR$(BUF(I,2)),2)
1710 V$=MID$(STR$(BUF(I,3)),2)
1720 W$=MID$(STR$(BUF(I,4)),2)
1730 PRINT #D, USING "LINE(&,&)-(&,&)":T $;U$,V$,W$
1740 RETURN
1750 REM •
1760 REM • Подпрограмма обработки прерываний
1770 REM •
1780 SPRITE OFF
1790 SWAP P,Q
1800 GOSUB 60
1810 FOR D=1 TO 100 : NEXT D
1820 SPRITE ON
1830 RETURN
1840 DATA 70,88,98,A8,C8,88,70,0
1850 DATA 20,60,20,20,20,20,70,0
1860 DATA 70,88,08,30,40,80,F8,0
1870 DATA F8,08,10,30,08,88,70,0
1880 DATA 10,20,50,90,F8,10,10,0
1890 DATA F8,80,F0,08,08,88,70,0
1900 DATA 38,40,80,F0,88,88,70,0
1910 DATA F8,08,10,20,40,40,40,0
1920 DATA 70,88,88,70,88,88,70,0
1930 DATA 70,88,88,78,08,10,E0,0
1940 DATA 80,40,20,10,8,4,2,1
1950 DATA 40,E0,40,0,0,0,0,0
```

При формировании изображения вам могут понадобиться такие команды:

L	При нажатии L определяется начальная точка прямой. При повторном нажатии вычерчивается прямая.
P	Закрашивается участок экрана.
U	Отменяется последняя команда.
DEL	Уничтожается программа, запущенная из памяти, а экран очищается
ESC	Генерируется программа

Один из спрайтов задействуется как курсор, другой — в качестве маркера, отмечающего начало прямой, и еще шесть в качестве "координатных", чтобы отмечать текущие координаты курсора на экране. Когда курсор касается одного из "координатных" спрайтов, они пропадают. Это выполняется с использованием прерывания по ON SPRITE GOSUB.

ПОКАДРОВАЯ МУЛЬТИПЛИКАЦИЯ

Рассмотренные ранее способы "оживления" изображения были несколько статичными: в то время как положение объекта менялось, его форма всегда оставалась одной и той же. Имеется и другая возможность создать иллюзию движения — непрерывно изменять форму спрайта, расположенного в определенной позиции.

При демонстрации кинофильмов пленка движется со скоростью 24 кадра в секунду. Имея 256 спрайтов и заменяя их через каждые 1/24 с, можно получить последовательность изображений, достаточную для имитации движения в течение чуть более 10 с. Если пользоваться большими спрайтами (которых всего 64), изображение будет более детальным, но время движения сократится до 2.5 с.

С помощью прерывания по ON INTERVAL легко имитировать временные интервалы длиной примерно 1/24 с для добавления новых изображений. В программе 9.7 показана простая последовательность, вызывающая иллюзию мультипликации.

Программа 9.7

```
10 REM •
20 REM • "Оживление" спрайта
30 REM •
40 COLOR 15,15,1
50 SCREEN 2,2
60 LINE (60,112)-(212,112),1
70 FOR I = 1 TO 4
80 S$=""
90 FOR J=1 TO 32:READ A:S$=S$+CHR$(A) :
NEXT
100 SPRITE$(I)=S$
110 NEXT
120 X=60 : SX=1 : I=1
130 PUT SPRITE 0,(X,96),1,I
140 X=X+SX:I=I+1
150 IF X<60 OR X>196 THEN SX=-SX
160 IF I>4 THEN I=1
170 FOR J=1 TO 25 : NEXT J
180 GOTO 130
190 DATA 0,2,5,5,2,19,15,3
200 DATA 3,3,2,2,2,4,4,6
```

```
210 DATA 0,0,0,0,0,128,64,32
220 DATA 0,16,240,0,0,0,0,0
230 DATA 0,2,5,5,2,3,7,11
240 DATA 19,3,2,2,2,2,2,3
250 DATA 0,0,0, 0,0,224, 0,0
260 DATA 0,0,128,64,40,16,0,0
270 DATA 0,2,5,5,2,3,7,11
280 DATA 11,11,2,2,2,2,2,3
290 DATA 0,0,0,0,0,32,192,0
300 DATA 0,0,128,64,64,64,96,0
310 DATA 0,2, 5, 5, 2, 3, 7, 11
320 DATA 19,19,2,2,2,3,2,3
330 DATA 0,0,0,0,64,64,192,0
340 DATA 0,0,128,64,128,0,128,0
```

ВИДЕОПАМЯТЬ

Помимо той области памяти, где хранятся интерпретатор MSX-Бейсика, программы и переменные, каждый MSX компьютер располагает памятью объемом в 16 К, предназначенной для работы видеопроцессора. Это память с очень быстрым доступом (т.е. процедуры чтения и записи требуют совсем немного времени). В ней содержится информация, необходимая для обеспечения нормального изображения на экране дисплея. Информация хранится в виде таблиц, причем каждая таблица определяет те или иные качества изображения.

В настоящем разделе будут рассмотрены способы обращения к видеопамяти (VRAM) с подробным анализом ее содержимого применительно к каждому типу экрана.

Чтение и запись. Видеопамять состоит из 16384 восьмибитных ячеек с адресами от 0 до 16383. Значения, хранящиеся в этих ячейках, можно прочесть с помощью функции VPEEK( ). В качестве аргумента задается адрес соответствующей ячейки видеопамяти, а возвращается ее текущее содержимое.

Дополнительной командой, осуществляющей запись в ячейки видеопамяти, является VPOKE. Записываемая величина должна быть целым числом от 0 до 255.

Текстовый экран 40 * 24. Среди всех типов экранов, если речь заходит о видеопамяти, тип SCREEN 0 требует меньше всего ресурсов. При работе в этом режиме выделяется всего 4 К видеопамяти, из которых реально используется лишь объем в 3008 байт. Он включает две таблицы: ***имена шаблонов*** и ***генератор шаблонов***.

Таблица имен шаблонов имеет длину 960 байт и начинается с адреса 0000. Данный текстовый экран насчитывает всего 960 позиций, куда можно помещать символы. Каждому знако-месту соответствует байт в таблице имен: верхнему левому углу экрана – байт 0000, верхнему правому – байт 0039, а нижнему правому – байт 0959. Запись в таблице имен содержит код шаблона, расположенного в определенный момент времени в соответствующем месте экрана. К примеру, если в левом

верхнем углу изображена буква "A", то в байте 0000 записано число 65 – именно тот номер кода, которым обозначается буква "A".

В генераторе шаблонов хранятся описания всех 256 шаблонов, которые можно изобразить на экране. Поскольку для одного шаблона требуется восемь байт (как в случае спрайтов 8 * 8), длина таблицы составляет 2048 байт. Таблица начинается с адреса 2048.

Содержимое таблицы имен шаблонов связано с расположением шаблонов в генераторе. Начальный адрес описания шаблона в видеопамяти определяется как произведение номера из таблицы имен на восемь (напомним, что описание одного шаблона занимает восемь байт), сложенное с 2048 (начальный адрес генератора шаблонов). Взяв для примера номер шаблона 65, вычислим начальный адрес этого шаблона:

2048 + (8 * 65) = 2568

Двоичное представление описания шаблона для "A" (ячейки с адресами с 2568 по 2575) таково:

```
00100000
01010000
10001000
10001000
11111000
10001000
10001000
00000000
```

Одна из операций, которые можно выполнять с такими таблицами,– изменение их содержимого. В программе 9.8 все буквы алфавита как заглавные, так и строчные, переопределяются с целью получения набора очень мелких букв. Это новое описание вносится в генератор с помощью функции VPOKE. Для данного режима экрана значимыми являются только шесть старших бит из описания шаблона.

Программа 9.8

```
10 REM *
20 REM * Мелкие буквы
30 REM *
40 SCREEN 0 : KEY OFF : PLAY"T255L6404"
50 PRINT "Загрузка данных" : PRINT
60 FOR I = 2568 TO 2775
70 READ A$ : VPOKE I,VAL("&H"+A$)
80 VPOKE I+256,VAL("&H"+A$)
90 NEXT I
100 PLAY "CD"
110 REM *
120 REM * Буквы алфавита
130 REM *
140 DATA 00,00,70,88,F8,88,88,00:' A
```

```
150 DATA 00,00,F0,48,70,48,F0,00:' B
160 DATA 00,00,78,80,80,80,78,00:' C
170 DATA 00,00,F0,88,88,88,F0,00:' D
180 DATA 00,00,F0,80,E0,80,F0,00:' E
190 DATA 00,00,F0,80,E0,80,80,00:' F
200 DATA 00,00,78,80,B8,88,70,00:' G
210 DATA 00,00,88,88,F8,88,88,00:' H
220 DATA 00,00,70,20,20,20,70,00:' I
230 DATA 00,00,70,20,20,A0,E0,00:' J
240 DATA 00,00,90,A0,C0,A0,90,00:' K
250 DATA 00,00,80,80,80,80,F8,00:' L
260 DATA 00,00,88,D8,A8,88,88,00:' M
270 DATA 00,00,88,C8,A8,98,88,00:' N
280 DATA 00,00,F8,88,88,88,F8,00:' O
290 DATA 00,00,F0,88,F0,80,80,00:' P
300 DATA 00,00,F8,88,A8,90,E8,00:' Q
310 DATA 00,00,F8,88,F8,A0,90,00:' R
320 DATA 00,00,78,80.70,08,F0,00:' S
330 DATA 00,00,F8,20,20,20,20,00:' T
340 DATA 00,00,88,88,88,88,70,00:' U
350 DATA 00,00,88,88,90,A0,40,00:' V
360 DATA 00,00,88,88,A8,D8,88,00:' W
370 DATA 00,00,88,50,20,50,88,00:' X
380 DATA 00,00,88,50,20,20,20,00:' Y
390 DATA 00,00,F8,10,20,40,F8,00:' Z
```

Программа 9.9 генерирует из звездочек буквы огромных размеров. Вначале считывается содержимое набора ячеек из таблицы имен, а затем с помощью подходящих шаблонов конструируются увеличенные буквы.

Программа 9.9

```
10 REM *
20 REM * Крупный заголовок
30 REM *
40 CLS
50 DIM A(80)
60 SCREEN 0 : KEY OFF : WIDTH 40
70 PRINT "Печать с помощью 20 символов"
80 FOR I=1 TO 20
90 A$=INPUT$(1)
100 PRINT A$;
110 NEXT
120 PRINT
130 REM *
140 REM * Коды для перевода строки
150 REM *
160 LPRINT CHR$(27);"3";CHR$(18);
```

```
170 REM •
180 REM • Основной цикл печати
190 REM •
200 FOR L=1 TO 20
210 X=2048+(VPEEK(39+L)*8)
220 FOR I=7 TO 2 STEP -1
230 FOR J=7 TO 0 STEP -1
240 IF (VPEEK(X+(J)) AND (2^I))<>0 THEN
LPRINT "•"; ELSE LPRINT SPACE$(1):
250 NEXT J
260 LPRINT
270 NEXT I
280 NEXT L
```

Текстовый экран 32 * 24. В режиме SCREEN 1 требуется больший объем видеопамяти, поскольку здесь применяются спрайты. Таблица имен и генератор шаблонов выполняют здесь те же функции, что и ранее, но располагаются начиная соответственно с адресов 6144 и 0000. В этом режиме можно шире использовать цвет. 32 байта видеопамяти отводятся под ***таблицу цветов***.

Генератор шаблонов можно разбить на 32 группы по восемь описаний шаблонов в каждой, т.е. шаблоны 0–7, 8–15 и т.д. Каждому такому блоку соответствует запись в таблице цветов. Старшие четыре бита определяют цвет основного изображения, а младшие четыре — цвет фона.

Начальный адрес таблицы цветов — 8192. В программе 9.10 показано, как подобная таблица позволяет изменять цвета изображения на экране.

Программа 9.10

```
10 REM •
20 REM • Изменение цвета
30 REM •
40 SCREEN 1
50 COLOR 15,4,4
60 FOR I=0 TO 255:PRINT CHR$(I)::NEXT
70 FOR I=0 TO 255
80 FOR J=8192 TO 8223
90 VPOKE J,I
100 FOR K=1 TO 25:NEXT K
110 NEXT J
120 NEXT I
```

Следует упомянуть в заключение, что в данном режиме имеют смысл все восемь бит описания шаблонов.

Экран с низким разрешением. Здесь мы вновь встречаемся с таблицей имен и генератором шаблонов, однако их назначение определяется несколько иначе. На этот раз таблица имен обращается к двум байтам генератора, где указывается цвет четырех пикселов экрана. Взаимосвязь

содержимого генератора с экраном схематически представлена на рис. 9.4.

Откровенно говоря, реализацию работы с экраном данного типа в MSX-Бейсике трудно назвать изящной и элегантной, и вмешательство в содержимое видеопамяти в данном случае вряд ли оправдано. Программа 9.11 заполняет экран черными и белыми клетками в шахматном порядке.

Программа 9.11

```
10 REM *
20 REM * Шахматная доска
30 REM *
40 COLOR 15,1,1
50 CLS
60 SCREEN 3
70 FOR I = 0 TO 7 STEP 2
80 VPOKE I,241 : VPOKE I+1,31
90 NEXT
100 FOR I=2048 TO 2815
110 VPOKE I,0
120 NEXT
130 GOTO 130
```

Экран с высоким разрешением. Для экрана этого типа как генератор шаблонов, так и таблица цветов насчитывают по 6144 байта – по одному байту на каждый из возможных блоков экрана размером 8 * 1.

Таблица цветов определяет цвет следующим образом. Пусть содержимому генератора

00110011

соответствует такое значение в таблице цветов:

11110001

Тогда шаблон будет иметь вид (Б – белый цвет, Ч – черный цвет):

ЧЧББЧЧББ

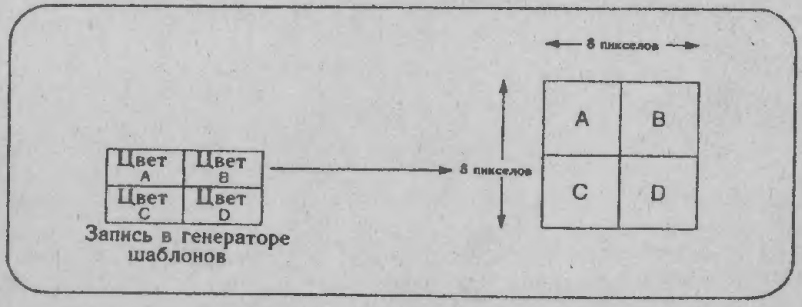

Рис. 9.4. Экран с низким разрешением и генератор шаблонов

Если значение генератора содержит единицу, то цвет пиксела определяется четверкой старших битов значения таблицы цветов. Нулевой бит генератора означает, что цвет пиксела задает младший полубайт.

Соответствие между именем шаблона и описанием в генераторе не настолько непосредственно, как раньше. Экран разделяется по горизонтали на три части.

1. Шаблоны с 0 по 255 из верхней трети экрана находятся в первой трети генератора.
2. Шаблоны с 0 по 255 из средней трети экрана находятся во второй трети генератора.
3. Шаблоны с 0 по 255 из нижней трети экрана находятся в последней трети генератора.

Таким образом, описание шаблона с данным номером зависит от местонахождения этого шаблона на экране.

Поскольку в Бейсике имеются разнообразные средства поддержки работы в таком режиме, манипуляции с подобными таблицами не так уж трудоемки. Но вот одно из очевидных приложений этих таблиц связано с заполнением экрана одинаковыми шаблонами (программа 9.12), и оно выполняется гораздо быстрее, чем с помощью ряда операторов PSET.

Программа 9.12

```
10 REM *
20 REM * Одинаковые шаблоны
30 REM *
40 COLOR 15,1,1
50 CLS
60 SCREEN 2
70 COLOR 15,1:CLS
80 FOR I = 0 TO 7
90 READ A : VPOKE I,A:VPOKE I+2048,A:VPOKE I+4096,A
100 NEXT
110 FOR I=6144 TO 6144+767
120 VPOKE I,0
130 NEXT
140 FOR I=8192 TO 14435
150 VPOKE I,241
160 NEXT
170 GOTO 170
180 DATA &B11111111
190 DATA &B10011001
200 DATA &B10111101
210 DATA &B11100111
220 DATA &B11100111
230 DATA &B10111101
240 DATA &B10011001
250 DATA &B11111111
```

Спрайты. Для каждого типа экрана, допускающего использование спрайтов, в видеопамяти резервируется по две области. Первая из них – ***генератор спрайтов***, где для определения каждого спрайта отводится в зависимости от его размера 32 или 8 байт. Функции генератора спрайтов те же, что и у других генераторов.

Гораздо более интересна другая область – ***таблица атрибутов спрайтов***. Каждая запись ее содержит четыре байта, функции которых показаны на рис. 9.5. Первый и второй байты определяют текущие координаты спрайта: по вертикали (Y) и по горизонтали (X). Таким образом, в любой момент программист может получить абсолютный адрес положения любого спрайта, что особенно полезно при работе с относительными указателями координат в операторе PUT SPRITE. Третий байт связан с текущим шаблоном спрайта. Он представляет собой число от 0 до 255 (для спрайтов 16 * 16 – от 0 до 63), по которому можно установить начальный адрес описания шаблона.

Рассмотрим четвертый байт содержимого таблицы атрибутов. Его младшие четыре бита определяют текущий цвет спрайта. Старший бит этого байта называется ***синхронизирующим***. Если его значение равно "1", положение спрайта смещается влево на 32 пиксела, что позволяет "чудесным" образом создавать на экране спрайты, как показано в программе 9.13.

Программа 9.13

```
10 REM *
20 REM * Синхронизирующий бит
30 REM *
40 COLOR 1,15,15
50 SCREEN 1,3 : WIDTH 30
60 SPRITE$(0)=STRING$(32,255)
70 PUT SPRITE 0,(0,80),8,0
80 VPOKE 6915,&B10001000
90 LOCATE 2,8,0:PRINT "Нажмите клавишу"
100 A$=INPUT$(1)
110 VPOKE 6915,&B00001000
120 END
```

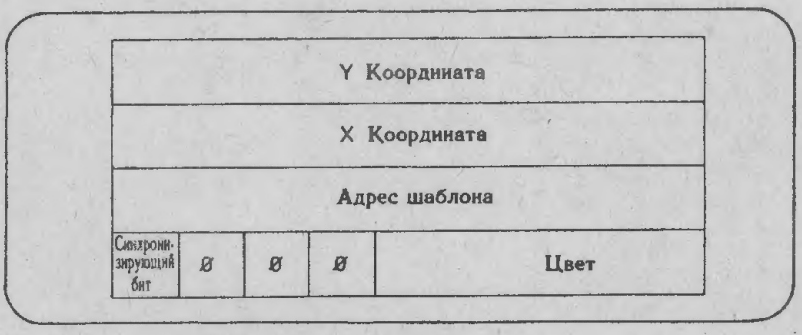

Рис. 9.5. Таблица атрибутов спрайтов

Рассмотрением спрайтов мы завершаем раздел, посвященный видеопамяти. Ее наиболее полезные свойства проявляются при работе в текстовых режимах, когда можно легко описать новые наборы символов. При работе с графическими экранами использование видеопамяти гораздо сложнее и требует навыка.

АДРЕСАЦИЯ С ПОМОЩЬЮ BASE

В имеющейся документации BASE описывается как функция, однако более логично рассматривать ее как переменную типа массива. Она устанавливает начальные (базовые) адреса для всех таблиц видеопамяти и для всех экранных режимов. В табл. 9.1 перечислены значения каждого элемента BASE и значения начальных адресов по умолчанию. Здесь приняты следующие обозначения:

Г – генератор шаблонов;
И – таблица имен;
Ц – таблица цветов;
А – таблица атрибутов спрайтов;
С – генератор спрайтов.

Указанные базовые адреса можно изменять новыми назначениями. Это следует делать крайне осторожно, иначе содержимое экрана можно полностью потерять. (В таком случае остается лишь перезагрузить компьютер.)

Таблица 9.1

	Адрес	Таблица
BASE(0)	0000	И SCREEN 0
BASE(2)	2048	Г SCREEN 0
BASE(5)	6144	И SCREEN 1
BASE(6)	8192	Ц SCREEN 1
BASE(7)	0000	Г SCREEN 1
BASE(8)	6912	А SCREEN 1
BASE(9)	14436	С SCREEN 1
BASE(10)	6144	И SCREEN 2
BASE(11)	8192	Ц SCREEN 2
BASE(12)	0000	Г SCREEN 2
BASE(13)	14436	А SCREEN 2
BASE(15)	2048	И SCREEN 3
BASE(17)	0000	Г SCREEN 3
BASE(18)	6912	А SCREEN 3
BASE(19)	14436	С SCREEN 3

Величины, которые можно устанавливать в качестве адресов для BASE( ), должны находиться в определенных границах. В табл. 9.2 даны предельные ограничения для таблиц видеопамяти.

Таблица 9.2

Таблица	Граница	Примеры базовых адресов
И (текст. 0)	1 K	1024,6144,7066
И (текст. 1)		
И (низ)		
И (выс)		
Г (текст. 0)	2 K	4096,6144
Г (текст. 1)		
Г (низ)		
С (любой тип)		
Ц (текст. 1)	64 байта	0000,128,512
Г (выс)	8 K	допустимы только 0000 и 8192
А (любой тип)	128 байт	2456,4024,14436

Наиболее целесообразно изменять базовый адрес генератора шаблонов, особенно в режиме SCREEN 0. Этому текстовому экрану требуется очень небольшой объем видеопамяти, и остается еще достаточно места для определения семи различных наборов символов. Используя полученные с помощью программы 9.8 мелкие буквы, но располагая их на этот раз с адреса 4096 (третий диапазон в 2 K), мы можем легко переходить на любой из двух алфавитов. Советуем перед выполнением программы 9.14 записать ее на ленту, чтобы не потерять содержимое экрана (на всякий случай).

Программа 9.14

```
10 REM •
20 REM • Мелкие и обычные буквы
30 REM •
40 SCREEN 0 : KEY OFF : PLAY"T255L64O4"
50 BASE(2)=2048
60 PRINT "Загрузка данных" : PRINT
70 FOR I = 4616 TO 4823
80 READ A$ : VPOKE I,VAL("&H"+A$)
90 VPOKE I+256,VAL("&H"+A$)
100 NEXT I
110 PLAY "CD"
120 PRINT "Нажмите клавишу для изменения"
130 PRINT "набора символов"
140 A$=INPUT$(1)
150 PLAY "L24O5CO6C"
160 BASE(2)=4096
170 REM •
180 REM • Буквы алфавита
```

```
190 REM *
200 DATA 00,00,70,88,F8,88,88,00:'A
210 DATA 00,00,F0,48,70,48,F0,00:'B
220 DATA 00,00,78,80,80,80,78,00:'C
230 DATA 00,00,F0,88,88,88,F0,00:'D
240 DATA 00,00,F0,80,E0,80,F0,00:'E
250 DATA 00,00,F0,80,E0,80,80,00:'F
260 DATA 00,00,78,80,B8,88,70,00:'G
270 DATA 00,00,88,88,F8,88,88,00:'H
280 DATA 00,00,70,20,20,20,70,00:'I
290 DATA 00,00,70,20,20,A0,E0,00:'J
300 DATA 00,00,90,A0,C0,A0,90,00:'K
310 DATA 00,00,80,80,80,80,F8,00:'L
320 DATA 00,00,88,D8,A8,88,88,00:'M
330 DATA 00,00,88,C8,A8,98,88,00:'N
340 DATA 00,00,F8,88,88,88,F8,00:'O
350 DATA 00,00.F0,88,F0,80,80,00:'P
360 DATA 00,00,F8,88,A8,90,E8,00:'Q
370 DATA 00,00,F8,88,F8,A0,90,00:'R
380 DATA 00,00,78,80,70,08,F0,00:'S
390 DATA 00,00,F8,20,20,20,20,00:'T
400 DATA 00,00,88,88,88,88,70,00:'U
410 DATA 00,00,88,88,90,A0,40,00:'V
420 DATA 00,00,88,88,A8,D8,88,00:'W
430 DATA 00,00,88,50,20,50,88,00:'X
440 DATA 00,00,88,50,20,20,20,00:'Y
450 DATA 00,00,F8,10,20,40,F8,00:'Z
```

ДРУГИЕ ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ ВИДЕОПАМЯТИ

Экономия памяти. Если в какой-то момент вы испытываете дефицит памяти, целые числа или коды ASCII можно поместить в видеопамять. Однако при таком методе необходимо постоянно пользоваться нулевым текстовым режимом экрана — при смене режима ваши данные будут стерты (см. программу 9.15). При этом освобождается объем памяти около 12 К, но заметно увеличивается время работы за счет выполнения операций VPEEK и VPOKE. Если скорость не является существенным фактором, то рассмотренный способ хранения данных в видеопамяти может быть весьма полезным, особенно для больших программ.

Программа 9.15

```
10 REM *
20 REM * Запись в режиме SCREEN 0
30 REM *
40 TP=4096: ' граница доступной памяти"
50 SCREEN 0:WIDTH 38
60 REM *
```

```
70 REM * Подпрограмма ввода
80 REM *
90 INPUT "Имя (* для окончания)":N$
100 IF LEFT$(N$,1)="*" THEN 210
110 IF TP+LEN(N$)+1>16383 THEN PRINT
"память заполнена": GOTO 210
120 FOR I=1 TO LEN(N$)
130 VPOKE TP,ASC(MID$(N$,I,1))
140 TP=TP+1
150 NEXT
160 VPOKE TP,0:TP=TP+1
170 GOTO 90
180 REM *
190 REM * Подпрограмма вывода
200 REM *
210 CLS
220 I=4096
230 X=VPEEK(I)
240 IF X=0 THEN PRINT : GOTO 260
250 PRINT CHR$(X);
260 I=I+1
270 IF I<TP THEN 230
280 END
```

Регистры видеопроцессора. Подобно звуковой микросхеме видеопроцессор также имеет ряд регистров, содержимое которых может быть считано или записано в них. Назначение каждого регистра показано на рис. 9.6.

Наибольший интерес представляют регистры 0, 1, 7 и регистр состояния, поскольку регистры 2–6 дублируют информацию, которую можно получить с помощью BASE( ).

Регистры 0 и 1. Биты регистра 1 имеют следующий смысл.

7 Определяет размер отведенной видеопамяти. Если он имеет нулевое значение, предполагается, что видеопамять занимает объем 4 К. Как правило, бит 7 содержит единицу, что соответствует объему видеопамяти в 16 К, требуемому для работы в графических режимах. Изменять содержимое этого бита, как правило, не нужно.

6 Применяется для выключения (0) и включения (1) экрана дисплея. Выключив дисплей, невидимый пользователю образ экрана можно вывести на печать или графопостроитель, а по окончании вывода дисплей можно вновь включить.

5 Единица означает разрешение прерываний, генерируемых видеопроцессором 50 раз в секунду; нуль соответствует запрещению прерываний. И этот бит нет особой нужды изменять.

4–3 Биты 4 и 3 в сочетании с первым битом нулевого регистра определяют текущий режим экрана. Каждому экрану соответствуют свои комбинации:

M1	M2	M3	Режим
0	0	0	текстовый 1
0	0	1	с высоким разрешением
0	1	0	с низким разрешением
1	0	0	текстовый 0

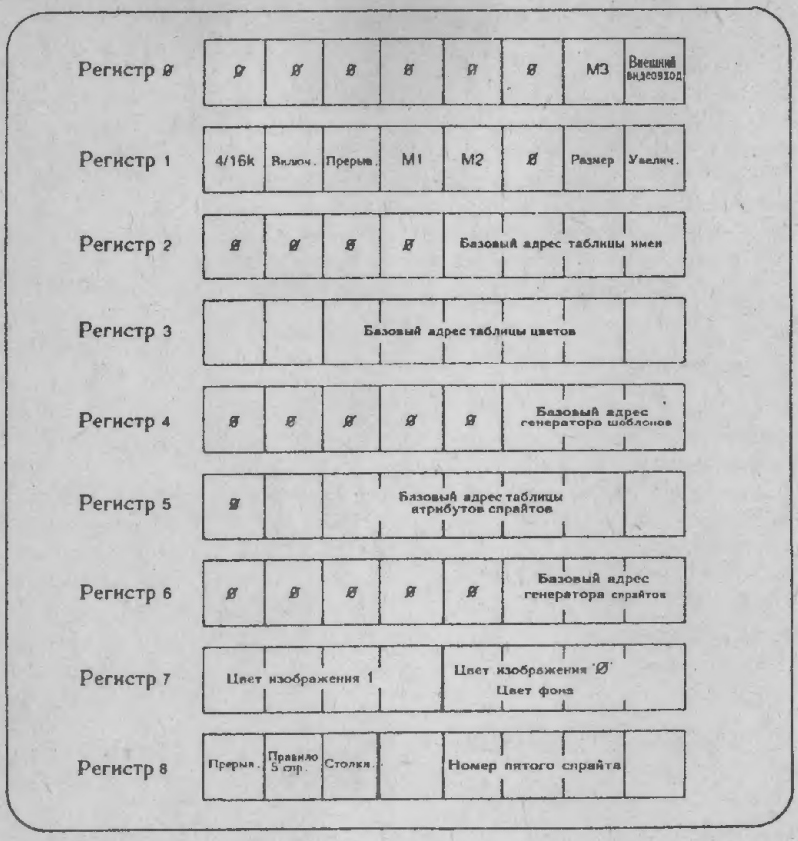

Рис. 9.6. Регистры видеопроцессора

1 Задает текущий размер спрайта. Единица соответствует спрайтам 16 • 16, нуль – спрайтам 8 • 8.

0 Единичное значение этого бита означает увеличение спрайтов.

Регистр 7. Назначение этого регистра – установка цветов основного изображения и фона для текстового режима 0. Верхний полубайт содержит код цвета основного изображения для этого экрана, а в нижнем полубайте хранится цвет фона для всех режимов. Если в текстовом режиме выполнить команду COLOR 15,4, в регистре 7 будет находиться число

11110100 (244).

Регистр 8 – регистр состояния. Этот регистр предназначен только для чтения. Его биты имеют следующий смысл:

7 Когда видеопроцессор генерирует сигнал прерывания, этот бит получает значение 1. При каждом чтении регистра состояния он сбрасывается. При программировании на Бейсике бит 7 фактически не используется.

6 Бит устанавливается в единицу всякий раз при нарушении правила пятого спрайта. Информация о состоянии бита 6 дает возможность управлять различными ситуациями, возникающими при появлении в строке пяти спрайтов и более.

5 Бит устанавливается при столкновении спрайтов, что позволяет программисту реагировать на столкновения без использования прерываний типа ON SPRITE GOSUB.

4–0 При нарушении правила пятого спрайта сюда записывается номер спрайта – "нарушителя".

Функция VDP. Эта функция позволяет прочитать содержимое любого регистра и записать информацию во все регистры, кроме восьмого. Перед записью в регистр лучше всего сначала считать его текущее содержимое, а затем с помощью логических команд AND и OR установить или сбросить нужные биты. Программа 9.16 показывает, как с помощью манипуляций с первым регистром (обращение к нему выглядит как VDP(1)) выполняется очистка экрана.

Программа 9.16

```
10 REM *
20 REM * Очистка экрана
30 REM *
40 COLOR 15,4,4
50 SCREEN 2
60 VDP(1)=VDP(1) AND &B10111111
70 CIRCLE (128,96),90,15:PAINT (128,96)
80 PLAY "c
90 VDP(1)=VDP(1) OR &B01000000
100 GOTO 100
```

Программа 9.17 дает возможность проследить роль регистра состояния (VDP(8)) в управлении столкновениями спрайтов.

Программа 9.17

```
10 REM *
20 REM * Управление столкновениями спрайтов
30 REM *
40 SCREEN 2,2
50 SPRITE$(0)=STRING$(32,255)
60 PUT SPRITE 0,(112,80),1,0
70 PUT SPRITE 1,(10,80),8,0
80 IF (VDP(8) AND 32) = 32 THEN GOTO 110
```

```
90 PUT SPRITE 1,STEP(1,0),8,0
100 GOTO 80
110 BEEP:GOTO 110
```

На этом мы заканчиваем описание работы видеопроцессора компьютеров MSX. Еще раз напоминаем, что при использовании регистров видеопроцессора и видеопамяти необходима предельная аккуратность. Если можно применить какие-то другие команды Бейсика, вероятно, так и следует поступать: программа будет легче читаться и станет удобнее в работе.

СВОДКА СЛУЖЕБНЫХ СЛОВ

Команды работы со спрайтами

SPRITE$ (<номер шаблона>)
PUT SPRITE <номер слоя>,<указатель координат>[,<цвет>]
[,<номер шаблона>]

Команды и функции видеопамяти

VPEEK (<адрес видеопамяти>)
VPOKE <адрес видеопамяти>,<записываемое число>
BASE (<индекс>)

Команды и функции видеопроцессора

VDP (<номер регистра>)

ПРИЛОЖЕНИЯ

1. ДОПОЛНИТЕЛЬНЫЕ ФУНКЦИИ

Это приложение дополняет список функций MSX-Бейсика. Приведем синтаксис дополнительных функций в обычных обозначениях с кратким описанием их назначения.

EXP (<X>) возвращает численное значение обычной экспоненциальной функции, т.е. e^x. Значение X не должно превышать 145.06286085862.

LOG (<X>) - дополнительная функция к EXP( ); возвращает значение натурального логарифма выражения.

SQR (X) возвращает значение квадратного корня из данного числового выражения.

PAD (N) считывает данные с графического планшета, подключенного к порту для джойстика. Значения порта A считываются при N, равном 0, 1, 2 или 3, а с порта B при N, равном 4, 5, 6 или 7.

Смысл возвращаемой величины для каждого значения N:

0 или 4	возвращается 0 или -1 в зависимости от того, есть прикосновение пера к планшету или нет;
1 или 5	возвращается координата X точки касания;
2 или 6	возвращается координата Y точки касания;
3 или 7	возвращается состояние кнопки-переключателя (-1, если кнопка нажата, иначе 0).

PDL (<N>) возвращает значение кода положения игрового рычажка. Для четных N считывается значение порта B, для нечетных — порта A. Возвращается целое число от 0 до 255.

PEEK (<адрес>) считывает значение любого указанного байта памяти. Адрес вводится в качестве аргумента. Добавление 65536 к отрицательному аргументу дает истинный адрес читаемого байта.

POKE <адрес>,<записываемая величина> записывает заданное значение в определенную ячейку памяти. Функцией не является, но сюда включен как дополнение к PEEK( ). Действует так же, как и VPEEK. Отметим, что видеопамять отделена от ОЗУ. POKE и PEEK не оказывают никакого воздействия на видеопамять.

VARPTR (<имя переменной>) возвращает адрес памяти, где хранится определенная переменная. Если возвращаемый адрес отрицателен, для получения истинного значения адреса к нему следует добавить 65536.

Для строковых данных VARPTR возвращает адрес, где хранится строка. Строки содержатся в другой области ОЗУ, нежели одиночные переменные. Переменной нужно предварительно присвоить определенное значение, и лишь после этого брать в качестве параметра для VARPTR( ).

USR [<номер подпрограммы>](<аргумент>) обеспечивает доступ к подпрограммам, записанным в машинных кодах. Более подробно эта функция рассматривается в приложении 6.

2. СООБЩЕНИЯ ОБ ОШИБКАХ

Код	Сообщение	Перевод
1.	NEXT without FOR	NEXT без FOR
2.	Syntax error	Синтаксическая ошибка
3.	RETURN without GOSUB	RETURN без GOSUB
4.	Out of DATA	Исчерпаны данные DATA
5.	Illegal function call	Неправильный вызов функции
6.	Overflow	Переполнение
7.	Out of memory	Выход за пределы памяти
8.	Undefined line number	Не определен номер строки
9.	Subscript out of range	Выход за пределы массива
10.	Redimensioned array	Повторное задание размерности массива
11.	Division by zero	Деление на нуль
12.	Illegal direct	Неправильная команда в режиме непосредственного выполнения
13.	Type mismatch	Несоответствие типа
14.	Out of string space	Исчерпано место для строковых переменных
15.	String too long	Слишком длинная строка
16.	String formula too complex	Строковая формула слишком сложна
17.	Can't continue	Невозможность продолжения
18.	Undefined user function	Неопределенная функция пользователя
19.	Device I/O error	Ошибка устройства ввода/вывода
20.	Verify error	Ошибка контроля
21.	No RESUME	Отсутствует оператор RESUME
22.	RESUME without error	Оператор RESUME при отсутствии ошибки
23.	Unprintable error	Сообщение по данной ошибке не может быть напечатано
24.	Missing operand	Пропущенный операнд
25.	Line buffer overflow	Переполнение буфера строки
26-49.	Unprintable error	Сообщение по данной ошибке не может быть напечатано
50.	Field overflow	Переполнение поля
51.	Internal error	Внутренняя ошибка
52.	Bad file number	Неправильный номер файла

53.	File not found	Файл не найден
54.	File already open	Файл уже открыт
55.	Input past end	Ввод после конца файла
56.	Bad file name	Неправильное имя файла
57.	Direct statement in file	Оператор прямого режима в файле
58.	Sequential I/O only	Только последовательный ввод/вывод
59.	File not OPEN	Файл не открыт с помощью команды OPEN
60.	Bad FAT	Неверная информация в FAT (таблице распределения файлов)
61.	Bad file mode	Неверный режим обращения к файлу
62.	Bad drive name	Неправильное имя диска
63.	Bad sector number	Неправильный номер сектора
64.	File still open	Файл все еще открыт
65.	File already exists	Файл уже существует
66.	Disk full	Диск заполнен
67.	Too many files	Слишком много файлов
68.	Disk write protected	Защита диска от записи
69.	Disk I/O error	Ошибка при дисковой операции ввода/вывода
70.	Disk offline	Диск в автономном режиме или выключен
71.	Rename across disk	Переименование с одного диска на другой
72-255.	Unprintable error	Сообщение по данной ошибке не может быть напечатано

[Коды 60–71 дополнены переводчиком.]

3. ДОПОЛНИТЕЛЬНЫЕ СЛУЖЕБНЫЕ СЛОВА

Служебными (зарезервированными) словами являются все имена операторов Бейсика и обозначения специальных функций. Имеется также ряд команд, которые интерпретатор опознает, но не может непосредственно выполнить. Они главным образом связаны с обработкой информации, хранящейся на дисках. Среди них выделяется служебное слово MAXFILES, которое фактически составлено из двух слов.

Дополнительные служебные слова:

ATTR$	CMD	COPY	DSKI$	DSKO$	FIELD
FILES	GET	IPL	KILL	LFILES	LSET
MAX	NAME	RSET	SET		

4. ТАБЛИЦЫ ЛОГИЧЕСКИХ ОПЕРАЦИЙ

Действие всех логических операторов описывается следующими таблицами:

NOT

X	NOT X
0	1
1	0

OR

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

EQV

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

AND

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

XOR

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

IMP

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

5. ТАБЛИЦА ЧАСТОТ

В этом приложении приведены значения частот, которые можно использовать в операторе SOUND, и соответствующие им значения команды N музыкального макроязыка.

N	PTH	PTH	N	PTH	PTH	N	PTH	PTH	N	PTH	PTH
0	85	0	25	39	3	49	202	0	73	50	0
1	156	12	26	250	2	50	190	0	74	48	0
2	231	11	27	207	2	51	180	0	75	45	0
3	60	11	28	167	2	52	170	0	76	42	0
4	155	10	29	129	2	53	160	0	77	40	0
5	2	10	30	93	2	54	151	0	78	38	0
6	115	9	31	59	2	55	143	0	79	36	0
7	235	8	32	27	2	56	135	0	80	34	0
8	107	8	33	253	1	57	27	0	81	32	0
9	242	7	34	224	1	58	120	0	82	30	0
10	128	7	35	197	1	59	113	0	83	28	0
11	20	7	36	172	1	60	107	0	84	27	0
12	175	6	37	148	1	61	101	0	85	25	0
13	78	6	38	125	1	62	95	0	86	24	0
14	244	5	39	104	1	63	90	0	87	22	0
15	158	5	40	83	1	64	85	0	88	21	0
16	78	5	41	64	1	65	80	0	89	20	0
17	1	5	42	46	1	66	76	0	90	19	0
18	186	4	43	29	1	67	71	0	91	18	0
19	118	4	44	13	1	68	67	0	92	17	0
20	54	4	45	254	0	69	64	0	93	16	0
21	249	3	46	240	0	70	60	0	94	15	0
22	192	3	47	227	0	71	57	0	95	14	0
23	138	3	48	214	0	72	53	0	96	13	0
24	87	3									

Здесь РТН и РГН - соответственно значения регистров точной и грубой настройки.

6. КАРТА ПАМЯТИ И ФУНКЦИЯ USR

Область памяти, отведенная для использования системой, фиксирована. Остальной объем памяти (за исключением ПЗУ) можно распределить между программами в машинных кодах и Бейсик-программами. Оператор CLEAR задает объем памяти, отведенной для MSX-Бейсика. Если выполнена команда

CLEAR 200,&H9FFF

то это означает, что Бейсик-программы, переменные и т.д. могут располагаться в пространстве от нижней границы памяти (в системах с ОЗУ объемом в 64 К это адрес &H&8000) по адрес &H9FFF включительно. Все оставшееся пространство (от &HA000 до &HF379) отводится для программ в машинных кодах.

Оператор DEFUSR определяет начальные адреса для программ в машинных кодах, которых может быть до 10. Программы нумеруются от 0 до 9. По умолчанию берется значение 0.

Оператор DEFUSR1 = &HA000 указывает, что начальным адресом программы с номером 1 является &HA000.

Программа в машинных кодах, будучи один раз описанной, может аналогичным образом использоваться и в качестве функции, например:.

<переменная> = USR <номер программы>(<аргумент>)

Значение переменной передается программе в машинных кодах. При вызове программы число байтов, занимаемое этим значением, однозначно связанное с его типом, заносится в регистр A микропроцессора Z80 и в ячейку памяти &HF663. Возможные значения:

2 целочисленные
4 вещественные с одинарной точностью
8 вещественные с двойной точностью
3 строковые

Адрес переменной хранится в регистре HL.

Для строк имеется небольшое отличие. Адрес, где записана строка, находится в регистре DE.

Если аргументом функции USR является нуль, в программу в машинных кодах ничего не переносится.

А .В. Гиглавый

КРАТКИЕ СВЕДЕНИЯ О ДОПОЛНИТЕЛЬНЫХ ВОЗМОЖНОСТЯХ MSX-БЕЙСИКА ДЛЯ КОМПЬЮТЕРОВ MSX-2

Существуют две версии Бейсика MSX:

* версия 1.0 (компьютер MSX);
* версия 2.0 (компьютер MSX-2).

Номер версии предъявляется при запуске интерпретатора Бейсика. Для определения номера версии из программы необходимо проверить ячейку с адресом 2D(H) (она содержит нуль, если используется версия 1.0). Все функции и операторы, реализованные в версии 1.0, сохранены также в версии 2.0. Некоторые функции и операторы версии 1.0 (например, SCREEN) расширены с учетом дополнительных возможностей графики компьютера MSX-2. Кроме того, в компьютерах MSX-2 добавлены новые операторы.

ФУНКЦИИ

BASE(N)

N=числовое выражение (0 - 19)

Возвращает адрес таблицы T в режиме SCREEN M (N = 5*M + T).

T	Таблица
0	PTN (Таблица названий шаблонов)
1	CT (Таблица цветов)
2	PGT (Таблица генератора шаблонов)
3	SAT (Таблица атрибутов спрайтов)
4	SGT (Таблица генератора спрайтов)

В MSX-2 N принимает значения от 0 до 49, M – от 0 до 8.
X = числовое выражение (от -32768 до 65535).

VDP(X)

X = числовое выражение (от 0 до 7)

Считывает содержимое регистра с указанным номером в формате FIX(X). В MSX-2 X принимает значения от 0 до 45.

ПСЕВДОПЕРЕМЕННЫЕ

BASE(N)=X

N = числовое выражение (от 0 до 19);
X = числовое выражение.

Задает размещение таблиц видео-ОЗУ в SCREEN 0–3; N = M*5+T.

VDP(N)=X

N = числовое выражение (от 0 до 7);
X = числовое выражение (от 0 до 255).

Позволяет записывать данные в регистры видеопроцессора. В MSX-2 N принимает значения от 0 до 7, от 9 до 24 или от 33 до 47.

ОПЕРАТОРЫ И КОМАНДЫ

CIRCLE [STEP] (X,Y), R [,C][,начало[,конец[,аспект]]]

X,Y,R = числовые выражения (от 0 до 65535);

C = числовое выражение (от 0 до 15 в режиме SCREEN 2, 3, 4, 5, 7; от 0 до 3 в SCREEN 6; от 0 до 255 в SCREEN 8; по умолчанию цвет текста);
начало, конец = числовое выражение (от -2*PI до +2*PI)
аспект = любое числовое выражение(практически имеют смысл значения от 1/260 до 260).

Рисует эллипс с центром в (X,Y), длинная ось которого равна R; <аспект> представляет собой отношение между вертикальной и горизонтальной осями; <начало> и <конец>, если они заданы, определяют концы дуги (в радианах). Те же параметры с отрицательными значениями позволяют построить сектор (значение -0 при этом не допускается; вместо него используется -.01). Режимы SCREEN 4–8 допускаются только для MSX-2.

COLOR [T] [,B] [Bo]

T, B, Bo = числовые выражения (используемые значения: от 0 до 15 в режимах SCREEN 0–5 и 7; от О до 3 в SCREEN 6; от 0 до 255 в SCREEN 8; в SCREEN 6 <Bo> может иметь значения от 0 до 31)

Переопределяет цвета текста, фона и границы в соответствии с указанными номерами цветов; необходим по меньшей мере один параметр; T переопределяет цвет текста в графических режимах. Режимы SCREEN 4–8 имеются только в MSX-2.

COLOR = (N, R, G, B) (только для MSX-2)

N = числовое выражение (от 0 до 15);
R, G, B = числовое выражение (от 0 до 7).

Переопределяет оттенки цветов в палитре N в режимах SCREEN 0–7.

COLOR = RESTORE (только для MSX-2)

Без параметров.

Обеспечивает считывание таблицы палитры после ее загрузки в видео-ОЗУ.

COLOR SPRITE$(NS) = C$ (только для MSX-2)

NS = числовое выражение (от 0 до 32);
C$ = символьное выражение (16 символов).

Присваивает символ каждой строке спрайта NS (в режимах SCREEN 4–8); битовое представление кода данного символа имеет следующее значение:

бит 7	сдвигает строку на 32 точки влево
бит 6	игнорирует приоритеты и совпадения: цвет двух накладывающихся линий задается операцией ИЛИ между цветами
бит 5	игнорирует совпадения
бит 4	не используется
бит 3-0	определяет цвет линии

COLOR SPRITE(NS) = значение (только для MSX-2)

NS = числовое выражение (от 0 до 32);
значение = числовое выражение (от 0 до 127).

Присваивает всем строкам спрайта NS значение, битовое представление которого имеет тот же смысл, что и в COLOR SRITE$(NS) =; однако бит 7 здесь недоступен.

COPY (X1,Y1)-(X2,Y2) [,стр-источник] TO (X3,Y3)
[,стр-приемник[,логика]]
COPY (X1,Y1)-(X2,Y2) [стр-источник] TO {массив/файл}
COPY {массив/файл} [,направление] TO (X3,Y3)
[,стр-приемник[,логика]]
COPY массив TO файл
COPY файл TO массив (все разновидности – только для MSX-2)

X1-Y3 = числовые выражения (от 0 до 65535; см. LINE);
стр-источник = номер страницы, из которой берутся данные (от 0 до максимального числа страниц; по умолчанию – активизированная страница);
стр-приемник = номер страницы, на которую будут копироваться данные (от 0 до максимального числа страниц; по умолчанию – активизированная страница);
массив = имя символьной переменной с определенной размерностью;
файл = символьное выражение, специфицирующее двоичный файл (см. оператор BLOAD);

направление = числовое выражение (от 0 до 3; по умолчанию – 0);

логика = зарезервированное слово (по умолчанию – PSET).

Позволяет указывать перемещения файлов в режимах SCREEN 5–8 между страницами/экранами, массивами и файлами (которые могут не быть открытыми); размер массива задается так:

INT((S*ABS(X1-X2)+1)*(ABS((Y1-Y2)+1)+7)/8)+4

S = 4 в SCREEN 5, 7, 8 и 2 в SCREEN 6

Направление перемещения блока определяется следующим образом:

0	верхний левый	-->	нижний правый
1	верхний правый	-->	нижний левый
2	нижний левый	-->	верхний правый
3	нижний правый	-->	верхний левый

Логика определяется посредством комбинирования цвета С точки с цветом экрана SC для получения нового цвета SC:

XOR	SC = (NOT C) AND SC OR C AND (NOT SC)
OR	SC = C OR SC
AND	SC = C AND SC
PSET	SC = C
PRESET	SC = NOT C

(к перечисленным словам может быть добавлен префикс T: в этом случае палитра 0 ничего не модифицирует)

COPY SCREEN [режим] (только для MSX-2)

режим = 0 или 1 (по умолчанию 0)

Преобразует кадр от внешнего видеоисточника в числовые данные с целью дальнейшего повторного использования; если режим = 0, одно поле преобразуется и копируется на указанную страницу; если режим = 1, два поля преобразуются и копируются на предыдущую страницу (номер указанной страницы должен быть нечетным).Для обеспечения возможности применения этого оператора необходим видеоинтерфейс в компьютере.

DRAW список

Параметр С в указанном операторе с последующим номером цвета позволяет учитывать возможности оператора COLOR в компьютере MSX-2.

GET DATE переменная [,A] (только для MSX-2)

переменная = имя символьной переменной

Присваивает значение даты переменной в формате "год/месяц/день"; вариант А возвращает дату, определенную для подачи сигнала.

GET TIME переменная [,A] (только для MSX-2)

Присваивает значение времени переменной в формате "час/минута/секунда"; вариант A возвращает время, определенное для подачи сигнала.

LINE [[STEP] (X1,Y1)] - [STEP] (X2,Y2) [[,C],{/BF}]

X1-Y1 = числовое выражение (от -32768 до 65535);
C = числовое выражение (от 0 до 15 в режимах SCREEN 2, 3, 4, 5, 7; от 0 до 3 в SCREEN 6; от 0 до 255 в SCREEN 8; по умолчанию – цвет текста).

Для данного оператора режимы SCREEN 4–8 предусмотрены только в компьютерах MSX-2.

LOCATE [C] [,L] [,K]

C = числовое выражение (значения: от 0 до WIDTH);
L = числовое выражение (значения: от 0 до 22 или 23);
K = числовое выражение (значения: от 0 до 1; по умолчанию – 0).

Помещает текстовый курсор в колонку C на строку L; определяет, должен появляться курсор (1) или нет (0). В компьютерах MSX параметр WIDTH изменяется в пределах от 1 до 40 в режиме SCREEN 0, и от 1 до 32 в SCREEN 1. В компьютерах MSX-2 параметр WIDTH изменяется в пределах от 1 до 80 в режиме SCREEN 0.

ON событие GOSUB список

событие = может быть задано одно из следующих событий:

INTERVAL = значение (от 1 до 65535; 50 = 1 секунда);
таймер

KEY:	нажатие функциональной клавиши
SPRITE:	столкновение спрайтов
STOP:	нажатие клавиш CTRL+STOP
STRIG:	нажатие кнопки триггера

В компьютерах MSX-2 выявление столкновения спрайтов может быть отменено. (См. подробнее описание операторов COLOR SPRITE$ и COLOR SPRITE.)

PAINT [STEP] (X,Y) [,C] [,BC]
X,Y = числовые выражения, возвращающие выводимые на экран координаты;
C,BC = числовые выражения (от 0 до 15 в режимах SCREEN 2–5 и 7; от 0 до 3 в SCREEN 6; от 0 до 255 в SCREEN 8).

Заполняет участок экрана, содержащий точку (X,Y), цветом C (по умолчанию – цветом текста); закрашенный участок обводится цветом BC (по умолчанию – C); в режимах SCREEN 2 и 4 BC не может отличаться от C. Экранные режимы SCREEN 4–8 предусмотрены только в компьютерах MSX-2.

PSET [STEP] (X,Y) [,C] и PRESET [STEP] (X,Y) [,C]

X,Y = числовые выражения (от -32768 до 65535);
C = числовое выражение (от 0 до 15 в режиме SCREEN 1–5 и 7; от 0 до 3 в SCREEN 6; от 0 до 255 в SCREEN 8).

Режимы SCREEN 4–8 предусмотрены только в компьютерах MSX-2.

PUT SPRITE NS [,STEP] (X,Y) [,C] [,NP]

NS = числовое выражение (от 0 до 31);
X,Y = числовые выражения (от -32768 до 65535; по умолчанию – последние координаты, присвоенные NS);
C = числовое выражение (от 0 до 15 в режимах SCREEN 0–5 и 7; от 0 до 3 в SCREEN 6; от 0 до 255 в SCREEN 8; по умолчанию – цвет текста или последний цвет, присвоенный NS посредством PUT SPRITE или COLOR SPRITE);
NP = числовое выражение (от 0 до 255 или от 0 до 63 в зависимости от размера; по умолчанию равно NS или последнему NP, присвоенному NS).

Помещает спрайт NS в точку с указанными координатами по модулю размера экрана и определяет цвет и шаблон; если задается цвет, то отменяется действие COLOR SPRITE$ или COLOR SPRITE. Режимы SCREEN 4–8, так же, как и операторы COLOR SPRITE($), предусмотрены только в компьютерах MSX-2. В MSX-2 на одной линии может быть выведено восемь спрайтов в режимах SCREEN 4–8. В SCREEN 1–3 на одной линии может быть выведено только четыре спрайта.

SCREEN [SM] [,SS] [,KC] [,BD] [,PO] [,DM]

SM = числовое выражение (режим) (от 0 до 8);
SS = числовое выражение (размер и увеличение спрайтов)(от 0 до 3);
KC = числовое выражение (звук клавиши) (от 0 до 255);
BD = числовые выражения (скорость вывода на кассету)(1 или2);
PO = числовое выражение (тип принтера) (от 0 до 255);
DM = числовое выражение (режим вывода) (от 0 до 3).

Задает режим экрана, размер и масштаб увеличения спрайтов; управляет эхо-сигналом при нажатии клавиш, скоростью передачи данных на "CAS:" и объявляет тип подключенного принтера; определяет режим вывода на экран. Если параметр пропущен, сохраняется предыдущее значение.
SM очищает экран;
SS стирает шаблон спрайта.
SM изменяется в пределах от 0 до 8 или от 0 до 6 в зависимости от размера видео-ОЗУ; по умолчанию равен 0 или 1.
SS: по умолчанию 0

0	матрица 8 x 8	масштаб 1
1	матрица 8 x 8	масштаб 2
2	матрица 16 x 16	масштаб 1
3	матрица 16 x 16	масштаб 2

KC: 0 - эхо-сигнала нет (по умолчанию < > 0)
BD: по умолчанию 1

1 1200 Бод
2 2400 Бод

PO: 0 - принтер MSX (по умолчанию)
DM: по умолчанию 0

0 обычный монитор
1 монитор с чересстрочной разверткой
2 обычный монитор:поочередный вывод
3 монитор с чересстрочной разверткой:
поочередный вывод

Режимы SCREEN (SM) 4–8 и параметр типа монитора (DM) предусмотрены только в компьютерах MSX-2.

SET вариант (эта группа операторов имеется только в компьютерах MSX-2)

вариант: см. ниже

Операторы SET обеспечивают начальные установки, результаты которых сохраняются в памяти даже при выключении питания; они изменяют значение, инициализирующее систему; варианты PASSWORD, TITLE и PROMPT не могут использоваться вместе, так как они хранятся в одной и той же области памяти. Другие варианты этого оператора действуют как обычные операторы.

SET ADJUST (X,Y)

X,Y = числовые выражения (от -7 до 8)

Центрирует изображение на мониторе.

SET BEEP звук,громкость

звук = числовое выражение (от 1 до 4);
громкость = числовое выражение (от 1 до 4).

Устанавливает звук и громкость при выполнении оператора BEEP.

SET DATE дата [,A]

дата = дата в формате "ГГ/ММ/ДД".

Устанавливает дату во внутреннем таймере с батарейным питанием; наличие A определяет дату подачи сигнала.

SET PAGE выведенная, активная

выведенная = номер страницы, выведенной на экран;
активная = номер страницы, для которой возможна запись.

Обеспечивает одновременное использование нескольких страниц-экранов в режимах SCREEN 5–8; номера страниц зависят от режима и объема видео-ОЗУ:

Имеющиеся экранные режимы	Объем видео-ОЗУ 64 КБ	128 КБ
5	0 - 1	0 - 3
6	0 - 1	0 - 3
7	---	0 - 1
8	---	0 - 1

SET PASSWORD текст

текст = строка (не более 255 символов).

Определяет пароль; это слово должно использоваться всякий раз при включении. Если оно забылось, нажмите одновременно клавиши GRAPH и STOP; для отмены его введите нулевую строку или лучше переопределите сообщение Ok.

SET PROMPT текст

текст = строка (максимум 6 символов).

Заменяет сообщение Ok другим коротким текстом.

SET SCREEN

Без параметров.

Сохраняет следующие параметры (значения во время выполнения):
экранный режим (от 0 до 1);
WIDTH (от 1 до 80);
цвет текста (от 0 до 15);
цвет фона (от 0 до 15);
цвет границы (от 0 до 15);
функциональные клавиши (ON или OFF);
звук при нажатии клавиши (ON или OFF);
принтер (MSX или другой тип);
скорость передачи (1200 или 2400 Бод);
режим вывода (от 0 до 3).

SET TIME время [,A]

время = время в следующем формате "ЧЧ:ММ:СС".

Устанавливает внутренние часы; если присутствует A, время будет считаться временем подачи сигнала; команда SET TIME должна вводиться перед SET DATE для установки даты и времени подачи сигнала.

SET TITLE текст [,цвет]

текст = строка (максимум 6 символов);
цвет = номер цвета (от 1 до 4).

Позволяет задать заголовок, выводимый на монитор при включении компьютера MSX-2 (под эмблемой MSX). Если заголовок содержит в точности шесть символов, экран с эмблемой и заголовком сохраняется до нажатия любой клавиши.

SET VIDEO режим [,YM ,CB [,SY [,AU [,VI [,AV]]]]]

режим = числовое выражение (от 0 до 3);
YM = числовое выражение (от 0 до 1);
CB = числовое выражение (от 0 до 1);
SY = числовое выражение (от 0 до 1);
AU = числовое выражение (от 0 до 3);
VI = числовое выражение (от 0 до 1);
AV = числовое выражение (от 0 до 1).

Осуществляет управление выводом на экран (использование внешнего источника видеосигнала и видеоинтерфейса).

режим: по умолчанию – 0; определяет источник видеосигнала

0 компьютер
1 компьютер
2 компьютер и внешний сигнал
3 внешний сигнал

YM: по умолчанию – 0; управляет интенсивностью изображения от источника видеосигнала

0 нормальная интенсивность
1 пониженная интенсивность

CB: по умолчанию - 0; определяет направление передачи данных о цветности

0 компьютер ---> экран
1 экран ---> компьютер

SY: по умолчанию - 0; определяет происхождение сигнала синхронизации

0 внутренняя синхронизация
1 внешняя синхронизация (если режим не равен 0)

AU: по умолчанию – 0; смешивает аудиосигнал компьютера и внешнего источника видеосигнала

0 компьютер
1 смешивание по правому каналу
2 смешивание по левому каналу
3 смешивание по обоим каналам

VI: указывает тип разъема для ввода внешнего видеосигнала

0 RGB
1 RCA

AV: состояние сигнала управления AV на разъеме RGB

0 OFF (выключено)
1 ON (включено)

SPRITE {ON/STOP/OFF}

Без параметров.

Разрешает (ON), откладывает (STOP) или запрещает (OFF) прерывания по столкновению спрайтов. В компьютерах MSX-2 выявление совпадения спрайтов может быть отменено. (См. COLOR SPRITE и COLOR SPRITE$.)

WIDTH ширина

ширина = числовое выражение (от 1 до 80 в SCREEN 0; от 1 до 32 в SCREEN 1)

Устанавливает ширину текста на экране; выбирает режим вывода символов в режиме SCREEN 0:

1 - 40 нормальные символы
41 - 80 символы, сжатые до половинной ширины

Сжатые символы имеются в SCREEN 0 только в компьютерах MSX-2.

ПРИМЕЧАНИЕ

В компьютерах серии "КУВТ Ямаха MSX", составляющих меньшую часть КУВТ данного семейства в учебных заведениях в СССР, установлен графический контроллер, отвечающий требованиям стандарта MSX-2. Однако в этих более ранних компьютерах использован MSX-Бейсик версии 1.0. Поэтому хотя перечисленные в настоящем приложении расширения Бейсика и не могут быть использованы в компьютерах первого варианта КУВТ, доступ к графическому контроллеру возможен на уровне машинных команд и применяется в некоторых программных продуктах для КУВТ. Переход к уровню машинных команд порождает оп-

ределенные отличия между текстами программ, работающих на компьютерах КУВТ образцов 1985 и 1987 годов; нарушается мобильность программ, гарантируемая стандартом MSX для программ на Бейсике. Поэтому пользователь должен знать, для какого варианта MSX-компьютера была изначально разработана интересующая его программа.

Научное издание

Грехем Блэнд

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ БЕЙСИК В СТАНДАРТЕ MSX

Книга одобрена на заседании секции редсовета по электронной обработке данных в экономике 28.10.86

Зав.редакцией К.В.Коробов
Редактор Н.К.Логинова
Худож.редактор Ю.И.Артюхов
Техн.редактор Г.В.Полякова
Корректор Г.В.Хлопцева
Обложка художника Безрученкова Л.Е.

ИБ №2391

Подписано в печать 27.12.88. Формат 60 x 88 1/16. Бум. офс. № 2. Гарнитура „Литературная". Печать офсетная. Усл. п. л. 12, 74. Усл. кр.-отт. 12,99. Уч.-изд. л. 11,56. Тираж 55 000 экз. Заказ 4. Цена 80 коп.

Издательство "Финансы и статистика", 101000, Москва, ул. Чернышевского, 7.

Книга подготовлена к печати на ППЭВМ в системе Астра-П, разработанной совместным отделом НИЦЭВТ-ГИВЦ Госкомиздата СССР.

Типография им. Котлякова издательства "Финансы и статистика" Госкомиздата СССР, 195273, Ленинград, ул. Руставели, 13.